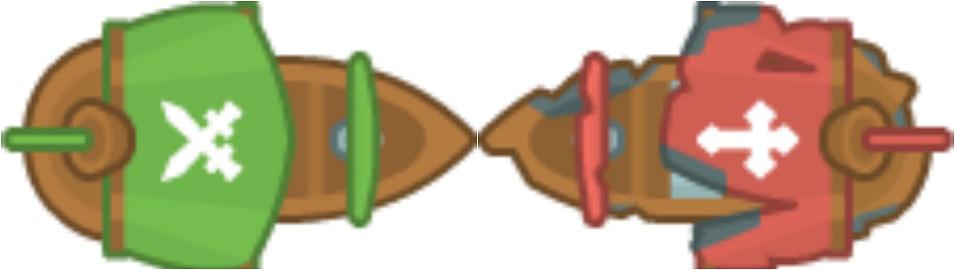
Faculty of Engineering

Software Project Report

**Simple Networked Game**

Ship Battle — *Sink the enemy players*



CENG 356 — Engineering Software Design

Fall 2017

David Li

V00818631

Term 3B

Computer Engineering

lidavid@uvic.ca

November 24, 2017

In partial fulfillment of the academic requirements of this course project

Creating a simple networked game — locally hosted with JavaScript

David Li

November 24, 2017

**Contents**

|  |  |
| --- | --- |
| [**List of Figures**](#page3) | **2** |
| [**List of Tables**](#page3) | **2** |
| [**Introduction**](#page4) | **3** |
| [Web Technologies](#page4) . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 3 |
| [Requirements](#page5) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 4 |
| [File Directory Layout](#page6) . . . . . . . . . . . . . . . . . . . . . . . . . . . | 5 |
| [Running a server using node.js](#page7) . . . . . . . . . . . . . . . . . . . . . | 6 |
| [**Game Logic**](#page7) | **6** |
| [Testing the game](#page9) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 8 |
| [Debugging the game](#page11) . . . . . . . . . . . . . . . . . . . . . . . . | 10 |
| [Game Object Properties](#page12) . . . . . . . . . . . . . . . . . . . . . . . . . | 11 |
| [**Player Logic**](#page13) | **12** |
| [Controls](#page17) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 16 |
| [Using Socket.io to Send/Receive Messages](#page18) . . . . . . . . . . . . . . . | 17 |
| [**Conclusion**](#page19) | **18** |
| [**Terms and abbreviations**](#page19) | **18** |
| [**References**](#page20) | **19** |
| [Resources Used](#page20) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 19 |
| [**Complete Code for Multiplayer Game**](#page21) | **20** |

|  |  |
| --- | --- |
| Creating a simple networked game — Using Javascript | 2 |

**List of Figures**

|  |  |  |
| --- | --- | --- |
| [1](#page4) | [Client Server Architecture](#page19) [Messaging Example](#page4) . . . . . . . . . . | 3 |
| [2](#page6) | [File Directory for Pirates Game displaying important files](#page6) . . . . | 5 |
| [3](#page7) | [Diagram for Client-Server Logic](#page7) . . . . . . . . . . . . . . . . . . | 6 |
| [4](#page8) | [Flow-Chart for Server Logic When Interacting with Players [](#page8)[3](#page20)[]](#page8) . . | 7 |
| [5](#page10) | [Loading Screen for game](#page10) . . . . . . . . . . . . . . . . . . . . . . | 9 |
| [6](#page10) | [Firing a Bullet in game](#page10) . . . . . . . . . . . . . . . . . . . . . . . | 9 |

* [An example of messages displayed server-side](#page11) . . . . . . . . . . 10

|  |  |  |
| --- | --- | --- |
| [8](#page13) | [Class Diagram for Pirates Game](#page13) . . . . . . . . . . . . . . . . . . | 12 |
| [9](#page14) | [Player is Disconnected from the Game.](#page14) . . . . . . . . . . . . . . | 13 |
| [10](#page14) | [Collision Boxes in the Game](#page14) . . . . . . . . . . . . . . . . . . . . | 13 |
| [11](#page15) | [Sequence Diagram of Typical Players Interacting with the Server](#page15) | 14 |

1. [Player loses when hit by bullet and disconnects from the game](#page16) . 15
2. [Flow Chart for Updating Hearts and Bullets Client-Side [](#page17)[3](#page20)[]](#page17) . . . . 16

|  |  |  |
| --- | --- | --- |
| [14](#page18) | [Logic To Spawn mini-ships Server-Side [](#page18)[3](#page20)[]](#page18) . . . . . . . . . . . . | 17 |
| [15](#page19) | [Player Sprite changing as health decreases](#page19) . . . . . . . . . . . . | 18 |

**List of Tables**

|  |  |  |
| --- | --- | --- |
| [1](#page5) | [Functional Requirements](#page5) . . . . . . . . . . . . . . . . . . . . . | 4 |
| [2](#page5) | [Non-Functional Requirements](#page5) . . . . . . . . . . . . . . . . . . . | 4 |
| [3](#page9) | [Results of testing game in di erent browsers](#page9) . . . . . . . . . . . | 8 |
| [4](#page12) | [Properties of Spawning Game Objects in Pirates Game](#page12) . . . . . | 11 |
| [5](#page12) | [Main Functions of **client.js**](#page12) . . . . . . . . . . . . . . . . . . . . . | 11 |

**Listings**

|  |  |  |
| --- | --- | --- |
| [1](#page18) | [Socket.io](#page20) [client and server commands to listen and receive data](#page18) | 17 |
| [2](#page21) | [Dependencies for game listed in package.json](#page21) . . . . . . . . . . | 20 |
| [3](#page22) | [HTML Code for the Pirates Game](#page22) . . . . . . . . . . . . . . . . . . | 21 |

* [Client-side logic](#page23) . . . . . . . . . . . . . . . . . . . . . . . . . . . 22

5 [Server-side logic](#page40) . . . . . . . . . . . . . . . . . . . . . . . . . . 39

**Abstract**

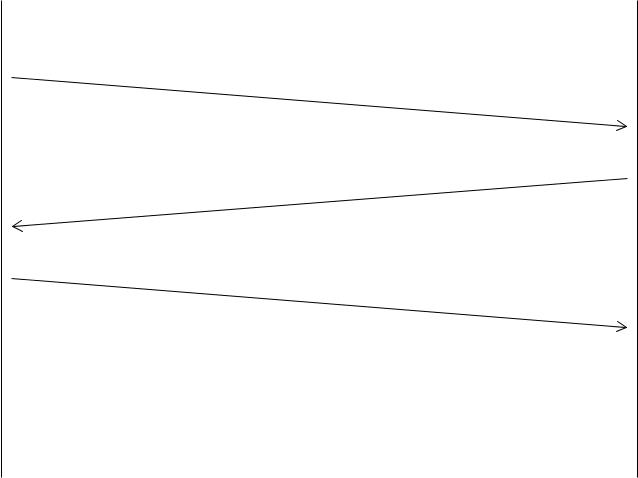
Documentation for a simple networked multi-player game that uses client-server architecture, describes the implementation and logic of the game.

|  |  |
| --- | --- |
| Creating a simple networked game — Using Javascript | 3 |

**Introduction**

Made in [JavaScript](#page20), in this simple multi-player game the player controls a ship and attempts to sink the other players and make them disconnect. AGILE so ware development methodology was used to produce this game.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | Client |  |  | Server |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
| Start | | | Requesting Access to Server | |  |  |  |
|  |  |  |  |  | Server receives | |  |
|  |  |  | Access Granted | | Server responds | |  |
|  |  |  |  |  |  |
| 2nd handshake begins | | | Acknowledgement | |  |  |  |
|  |  |  |  |  | Server receives | |  |



Send Data and Acknowledge



End

Figure 1: [Client Server Architecture](#page19) Messaging Example

**Web Technologies**

Major components of web technologies include HTML 5, CSS, and JavaScript. The client-server logic was created using [Socket.io](#page20) [[1](#page20)], and uses the Phaser Game Engine [[2](#page20)] to load sprites as well as game logic.

* [HTML](#page20) — structure for web content such as headings, paragraphs and forms.
* [CSS](#page19) — styling rules for [HTML](#page20) including background colours and fonts.
* [JavaScript](#page20) — programming language to dynamically update content, client-server logic, drawing graphics on web page (canvas) and run a web server. The main libraries used as listed below.

**–** Phaser — creating web games geared towards (2D arcade games).

**–** Socket.io — Used for the[Client Server Architecture](#page19).

**–** Node.js — allows[JavaScript](#page20)usageserver-side.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Creating a simple networked game — Using Javascript | | | | 4 | | |
| **Requirements** | |  |  |  |  |  |
|  |  | Table 1: Functional Requirements | |  |  |  |
|  | **Requirements** | **Description** |  |  |  |  |
|  | **Multi-player** | Ensure that one or more players can connect to the server | |  |  |  |
|  | **Movement** | A user can see their ship move across the map | |  |  |  |
|  | **Shooting** | Player can generate bullets. | |  |  |  |
|  | **Hearts** | User will restore health if ship collides with heart. | |  |  |  |
|  | **Mini-ships** | Obstacles will spawn and rotate in the game region. sys- | | |  | |
|  |  | tem. |  |  |  |  |
|  | **Bullets** | Mini-ships and Hearts are destroyed and players are dam- | | |  | |
|  |  | aged by this. |  |  |  |  |
|  | **Collisions** | Player will get pushed back if they get to close to another | |  |  |  |
|  |  | player |  |  |  |  |
|  | **Random Ships** | 1. User connects | 2. random ship is given |  |  |  |
|  |  |  | to user. |  |  |  |
|  | **Game Over** | **Success:** Other players can’t see user when health is zero. | |  |  |  |
|  |  | **Failure:** The user remains active in the game map. | |  |  |  |
|  |  | Table 2: Non-Functional Requirements | |  |  |  |
|  | **Requirements** | **Description** |  |  |  |  |
|  | **Enemy Ships** | The number of enemies is displayed on the screen. | |  |  |  |
|  | **Sound** | When a bullet is fired a sound will play. | |  |  |  |
|  | **Loading Screen** | The user will see how many files have loaded before the | | | | |
|  |  | game starts. |  |  |  |  |
|  | **Music** | When the player successfully joins the game, music plays. | | | | |

|  |  |
| --- | --- |
|  |  |
| **Number of Bullets** | Once the player runs out of players, they can reload. The |
|  | remaining number of bullets the player has is displayed on |
|  | the screen. |
| **Frame rate** | The server will update the positions of players 60 times a |
|  | second. |
| **Player Health** | Every ship in the game will have their health bar displayed |
|  | in the upper le corner. Player health is capped at 100 and |
|  | it updates when the player health changes. |

|  |  |
| --- | --- |
| Creating a simple networked game — Using Javascript | 5 |

**File Directory Layout**

system

 assets

 ship.png

 water.png

 cannonball.png

 js

 client.js

 healthMeter.js

 package.json

 index.html

 server.js

Figure 2: File Directory for Pirates Game displaying important files

|  |  |
| --- | --- |
| Creating a simple networked game — Using Javascript | 6 |

The most important files are the **client.js**, **server.js** and **index.html**, they im-plement the client-server architecture. Installing necessary packages is straight-forward by using the package manager for [Node.js](#page20), [NPM](#page20). Navigating to the main game directory and calling **npm install**, installs the required node modules ([Ex-press.js](#page19),[Socket.io](#page20)) specified in **package.json** ([2](#page6)) under the folder node\_modules.

**Running a server using node.js**

Running the game involves navigation to the main file directory [2](#page6), running node and opening localhost at port 8000.

Calling the following commands in the shell environment. **node** server.js. Then, the user can navigate to *http://localhost:5000/*.

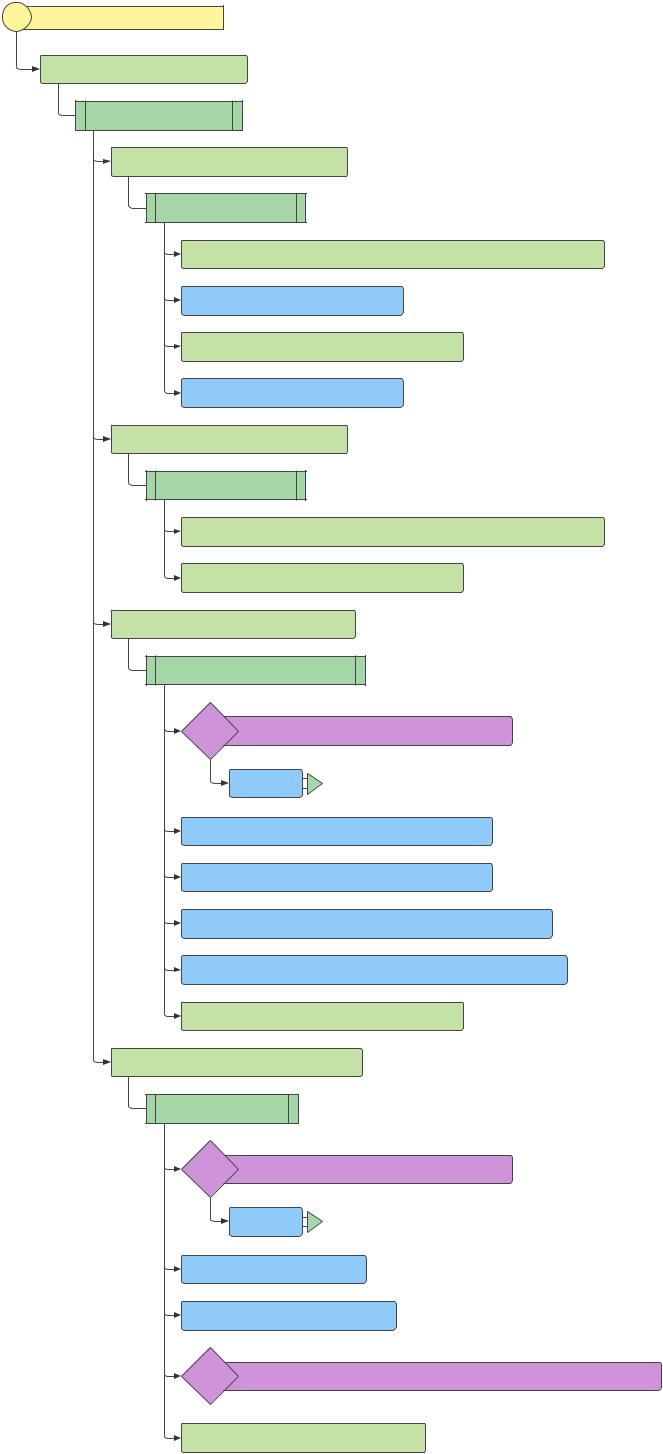
The server-client uses [Socket.io](#page20) [[1](#page20)] for real-time two way communication be-tween clients (browser) and the web server (localhost).

**Game Logic**

Figure 3: Diagram for Client-Server Logic

As shown in Figure [3](#page7), [Socket.io](#page20) relays messages from the client to the server and vice-versa. Information sent from the client-side (browser) include keyboard

|  |  |  |
| --- | --- | --- |
| Creating a simple networked game — Using Javascript | | 7 |
| and mouse input, player collisions with other players, and spawning bullets. | |  |
| Program: source module |  |  |
| io.on('connection', \*()) |  |  |
| function(socket) |  |  |
| socket.on('new-player', \*()) | |  |
| function(state) | |  |
| console.log('New player joined with state:', state... | |  |
| players[socket.id] = state | |  |
| io.emit('update-players', players) | |  |
| spawnObjectsAllowed = true | |  |
| socket.on('disconnect', \*()) | |  |
| function(state) | |  |
| console.log('Player disconnected with state: ', st... | |  |
| io.emit('update-players', players) | |  |
| socket.on('move-player', \*()) | |  |
| function(position\_data) | |  |
| if | (players[socket.id] == undefined) |  |
| + |  |  |
|  | return |  |
| players[socket.id].x = position\_data.x | |  |
| players[socket.id].y = position\_data.y | |  |
| players[socket.id].angle = position\_data.angle | |  |
| players[socket.id].health = position\_data.health | |  |
| io.emit('update-players', players) | |  |
| socket.on('shoot-bullet', \*()) | |  |
| function(data) | |  |
| if | (players[socket.id] == undefined) |  |
| + |  |  |
|  | return |  |
| var new\_bullet = data | |  |
| data.owner\_id = socket.id | |  |
| if | (Math.abs(data.speed\_x) > 20 || Math.abs(data.spee... |  |
| + |  |  |



bullet\_array.push(new\_bullet)

Figure 4: Flow-Chart for Server Logic When Interacting with Players [[3](#page20)]

|  |  |
| --- | --- |
| Creating a simple networked game — Using Javascript | 8 |

For this game, the server sends game assets (sprites, images and music) to the client, delivers information about the current location of players and obstacles, and manages player connecting/disconnecting to the server.

When a player connects, a new ship is creates and a message is sent to the server, then the servers sends another message and the player is allowed to roam around the game environment until more players connect.

Also, server waits until at one player is present before spawning hearts and mini-ships.

**Testing the game**

Overall, the game was tested using three modern commonly used browsers. No major errors were found, but Chrome runs the best, then Firefox and lastly Microso Edge.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Table 3: Results of testing game in di erent browsers | | |
| **Browser** | **Running** | **Running on Free Web** | **Other Notes** |
|  | **Locally** | **Server** |  |
| Chrome [[4](#page21)] | Runs well | Takes longer to load | Better than other browsers |
| Firefox [[5](#page21)] | Runs | Somewhat Slower than | A bit slow at times |
|  | adequately | chrome |  |
| Edge [[6](#page21)] | Bullets lag | Inadequate for in class | Loads fine. |
|  |  | demo |  |

In the upper le , the health bar for the player is shown, however, the condition of other ships is apparently with their sprites refer to Figure . When hearts or mini-ships collide with bullets they are destroyed and collisions with another player’s bullet damages that player.

Furthermore, the alpha value [[7](#page21)] of the damaged player is set to zero momentar-ily, and returns to 1 later. This should appear as flicking to the other players. The health of the player decreases. Whenever bullets are fired, or collisions occur, appropriate sound is played.

|  |  |
| --- | --- |
| Creating a simple networked game — Using Javascript | 9 |

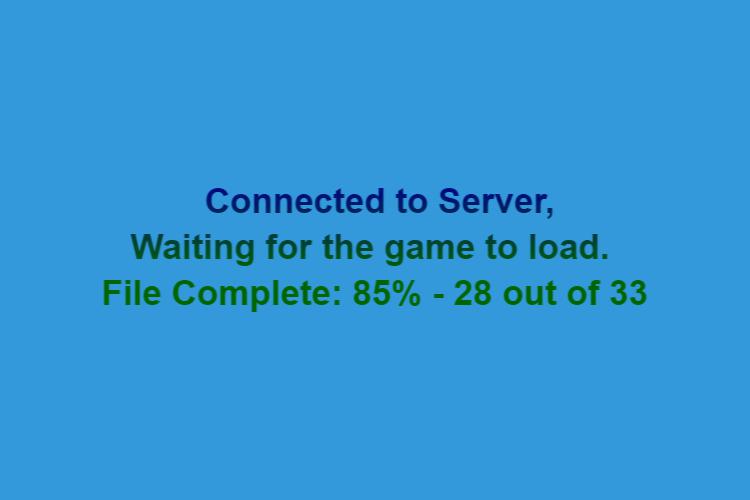


Figure 5: Loading Screen for game

Since music and sound files are used in the game, loading takes a few seconds, as a result it become necessary to add a loading screen to inform the player when they can start playing.

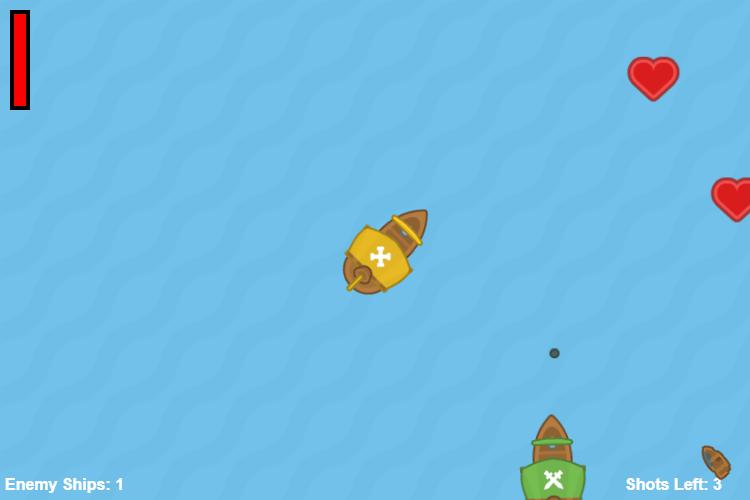


Figure 6: Firing a Bullet in game

When a player fires, a message is sent to the server to create a bullet and then it sends another message to update the location of all bullets for the clients.

|  |  |
| --- | --- |
| Creating a simple networked game — Using Javascript | 10 |

*Debugging the game*

Debugging this game requires validating both client-side and server-side logic simultaneously.This involves using the Chrome debug tools [[4](#page21)] and messages displayed on command prompt from [Node.js](#page20).

Figure 7: An example of messages displayed server-side

Interestingly, if the game failed to load, o en-times a syntax error was found in the **client.js** file. Likewise, when bullets and hearts were stationary, server disconnected because of a error in **server.js** and displayed an error message. In both cases, logging messages are incredibly helpful when debugging the game. Some problems encountered included:

* Uncertainty of how to implement [Client Server Architecture](#page19) and how orga-nize the code. For example, physics are computed client-side.
* Di erences between the information available for the server and client. One example is players receiving inaccurate information on the health of other players.

|  |  |
| --- | --- |
| Creating a simple networked game — Using Javascript | 11 |

* Another problem is simulating accurate collisions, but since [Phaser](#page20) is a game engine, it is pre-packaged with several options to compute physics.

**Game Object Properties**

The game objects spawned by the server are similar, bullets move at a constant relative speed of 17.5, the movement speed the x and y direction of hearts and mini-ships varies from -0.5 to 0.5. Only mini-ships rotate in-game and the direc-tion of rotation is randomized. When the Game Objects go outside the visible game region of they are destroyed.

Table 4: Properties of Spawning Game Objects in Pirates Game

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Property | Description | Bullet | Heart | Mini-Ship |
|  |  |  |  |  |
| x | Cartesian Coordinate (x-position) | [0*,* 1500] | [0*,* 1500] | [0*,* 1500] |
| y | Cartesian Coordinate (y-position) | [0*,* 1000] | [0*,* 1000] | [0*,* 1000] |
|  |  |  |  |  |
| speed*x* | Speed in the x direction | 17*.*5 | [−5*,* 5] | [−5*,* 5] |
| speed*y* | Speed in the y direction | 17*.*5 | [−5*,* 5] | [−5*,* 5] |
|  |  |  |  |  |
| rotation | Angular speed: CCW ( ) or CW( ) | — | — | *π/*200 |

Initially, the client loads all the assets, then creates the game environment and proceeds to update the game by sending/receiving messages from the server.

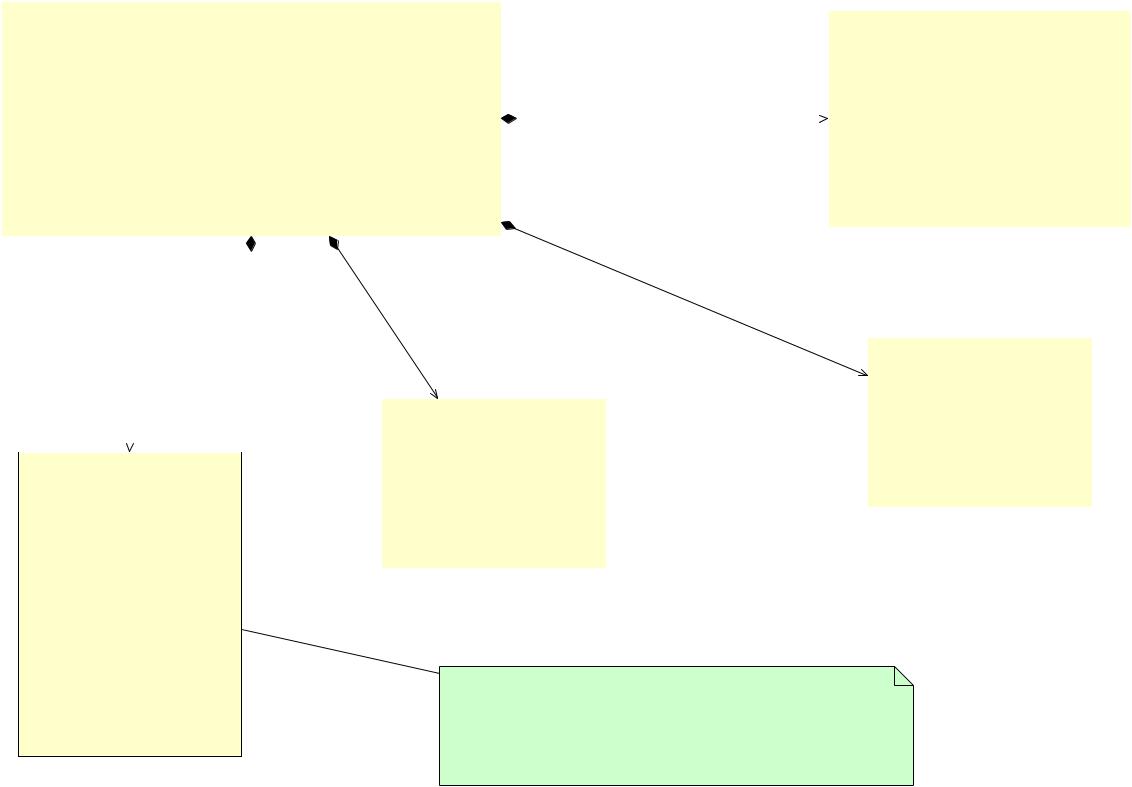
|  |  |
| --- | --- |
|  | Table 5: Main Functions of **client.js** |
| **Function** | **Description** |
| preload | Loads all the necessary sprites, music and sound e ects |
| create | A er loading is complete, spawn the player sprite, load the map |
|  | and display starting messages. |
| update | Move the camera to based on the player’s current location and |
|  | update position of other players. |

Although the game objects are created client-side, interactions between other game objects is handled by the server then updates are received client-side. (See Figures [8](#page13) [11](#page15) and [3](#page7)). Additionally, the positions and collisions of game objects with

|  |  |
| --- | --- |
| Creating a simple networked game — Using Javascript | 12 |

each other are handled server-side. This includes hearts, bullets, and mini-ships

* .



|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | Pirates Game | | | | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | Mini-ships | | | | | |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | WORLD SIZE : constant type = w:1500,h:1000 | | | | | | | | | | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  | + x : Number | | | | | | | |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | WINDOW WIDTH : constant type = 750 | | | | | | | | | | |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  | + y : Number | | | | | | | |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | server | |  |  |  |
|  | WINDOW HEIGHT : constant type = 500 | | | | | | | | | | | | |  |  |  |  |  |
|  |  |  |  |  | + speed | | x : Number | | | | | |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | spawns | | Mini-ships |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | + speed | | y : Number | | | | | |  |
|  | preload() : void | | | | | | |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  | + rotationDirection : Number | | | | | | | |  |
|  | create() : void | | | | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  | + rotation : Number | | | | | | | |  |
|  | GameLoop() : void | | | | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | CreateShip(string,Number,Number,Number) : Sprite | | | | | | | | | | | | |  |  | server | |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  | spawns | | | |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  | server | | | |  |  |  |  |  | destroys | | |  |  |  |  |  |  |  |
|  |  |  |  |  | server | | |  | spawns | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  | bullets | |  |  |  |  |  |  |  |  |  |  |  | Hearts | | | |  |  |
|  |  |  |  |  | creates | | |  |  |  |  |  |  |  |  |  |  | Hearts |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | + x : Number | | | | | |  |  |
|  | client | | | | player | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  | Bullet | | | | |  |  |  |  | + y : Number | | | | | |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | + speed | | | x : Number | | |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  | + x : Number | | | | |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | Player | | | | | |  |  |  |  |  |  | destroys |  |  | + speed | | | y : Number | | |  |  |
|  |  |  |  |  | + y : Number | | | | |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | + x : Number | | | | | | |  |  |  |  | + speed |  | x : Number | | |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  | + speed | | y : Number | | |  |  |  |  |  |  |  |  |  |  |  |  |
|  | + y : Number | | | | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | + rotation : Number | | | | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | + health : Number | | | | | | | player | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | + alive : Boolean | | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  | shoots | | | | bullets | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | + shot : Boolean | | | | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| + bullets : Number | | |  |  |
| + speed |  | x : Number |  |  |
| + speed |  | y : Number | Interactions between players and game objects is |  |
|  |  |
|  |  |  | controlled by the server and client. Each player |  |
| update () : void | | |  |
| updates their own position. |  |
|  |  |  |  |

Figure 8: Class Diagram for Pirates Game

**Player Logic**

In this game, the player controls a small pirate ship and attempts to defeat all enemies ships, while avoiding obstacles and collecting hearts to restore missing health. Shooting bullets at enemy players will damage them and when firing at hearts or mini-ships will destroy them.

* Note that Dri Wood is a term used interchangeably with mini-ships.

|  |  |
| --- | --- |
| Creating a simple networked game — Using Javascript | 13 |



Figure 9: Player is Disconnected from the Game.

The speed of the ship is linearly dependent on player health, in other words the more damaged the ship, the slower it moves. At zero health, the player cannot move at all. Simple bounding boxes are used for collisions.

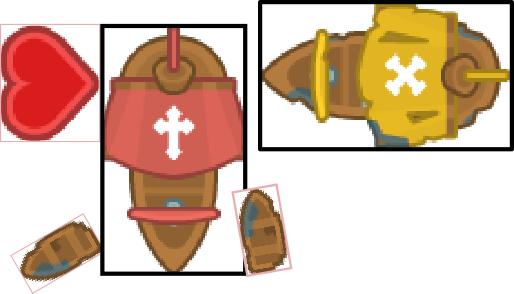
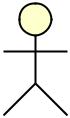
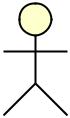


Figure 10: Collision Boxes in the Game

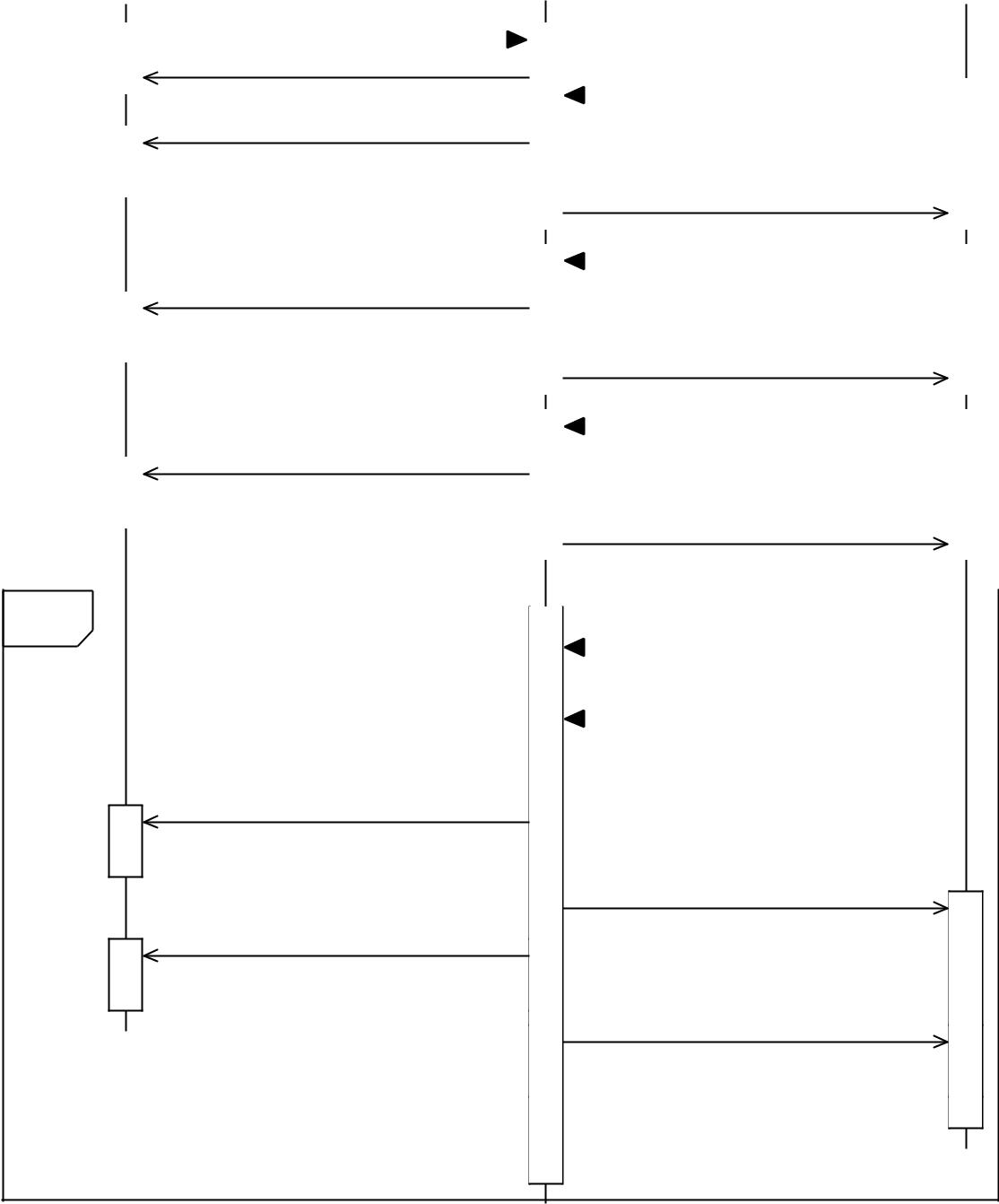
If the player collides with a bullet they take damage, colliding with a heart will restore health and remove that heart from the game, and colliding with ships will push the player back.

Furthermore, in order for the player to interact with other players or game objects on the map, the client must communicate with the server (See) [11](#page15).

|  |  |
| --- | --- |
| Creating a simple networked game — Using Javascript | 14 |



|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Player A: | | |  | Server: | | | |  | Player B: | | | |  |
|  |  |  | new-player |  | |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  | |  |  |  |  |  |  |  |  |
|  |  |  | update-players | |  |  |  |  | new-player |  |  |  |  |
|  |  |  | update-players | | |  |  |  |  |  | |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  | update-players |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  | |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  | bullets-update | | |  |  |  | shoot-bullet |  |  | |  |
|  |  |  |  |  |  |  |  | |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  | bullets-update |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  | |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  | update-players | | |  |  |  | move-player |  |  | |  |
|  |  |  |  |  |  |  |  | |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  | update-players |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  | |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| loop | | |  |  |  |  |  |  | ServerGameLoop | | | |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  | spawnHealthPacks | | | |  |
|  |  |  |  |  |  |  |  |  |  |



healthPack-update

healthPack-update

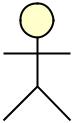
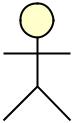
bullet-update

bullet-update

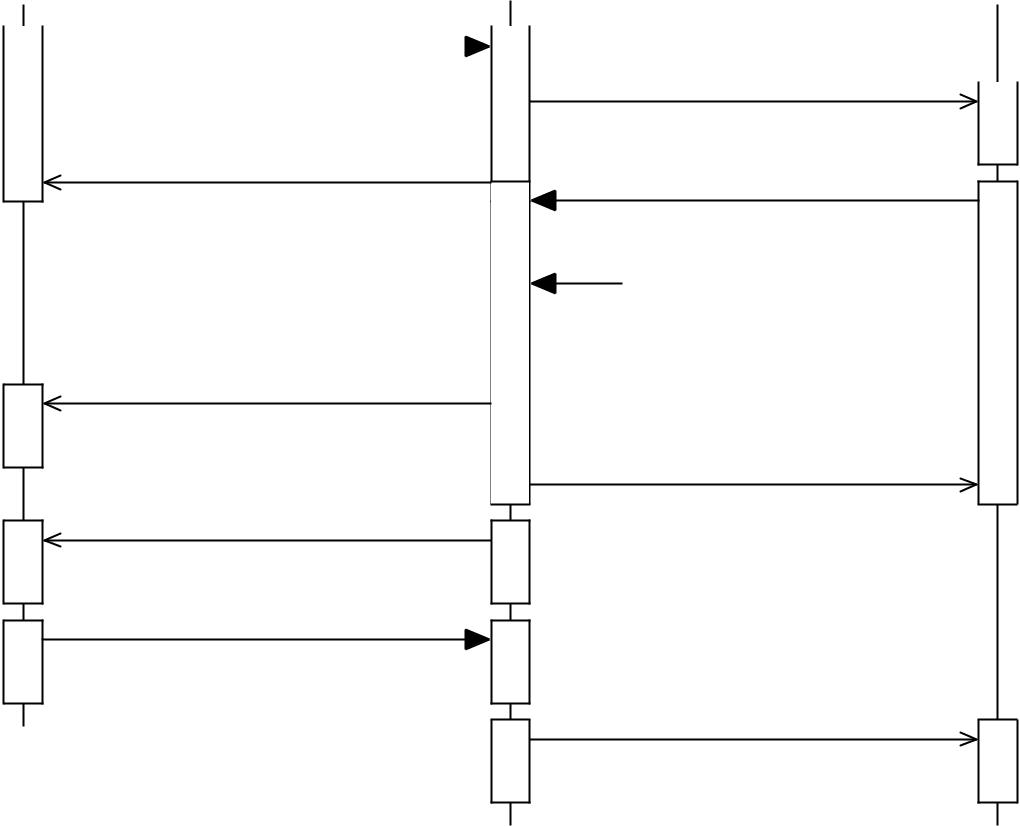
Figure 11: Sequence Diagram of Typical Players Interacting with the Server

|  |  |
| --- | --- |
| Creating a simple networked game — Using Javascript | 15 |

Whenever the player moves, shoots bullets and takes damage, the clients sends a message to the server and then updated information is sent to all the clients. In addition, the server periodically spawns obstacles in the game, and updates the positions of game objects see [11](#page15).



|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Player A: | | | |  |  |  |  |  |  |  | Player B: | | |  |
|  | Server: | | | |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  | move-player |  | |  |  |  |  | update-players |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |



update-players

bullets-update

player-hit

disconnect

shoot-bullet

 ServerGameLoop

bullets-update

update-players

Figure 12: Player loses when hit by bullet and disconnects from the game

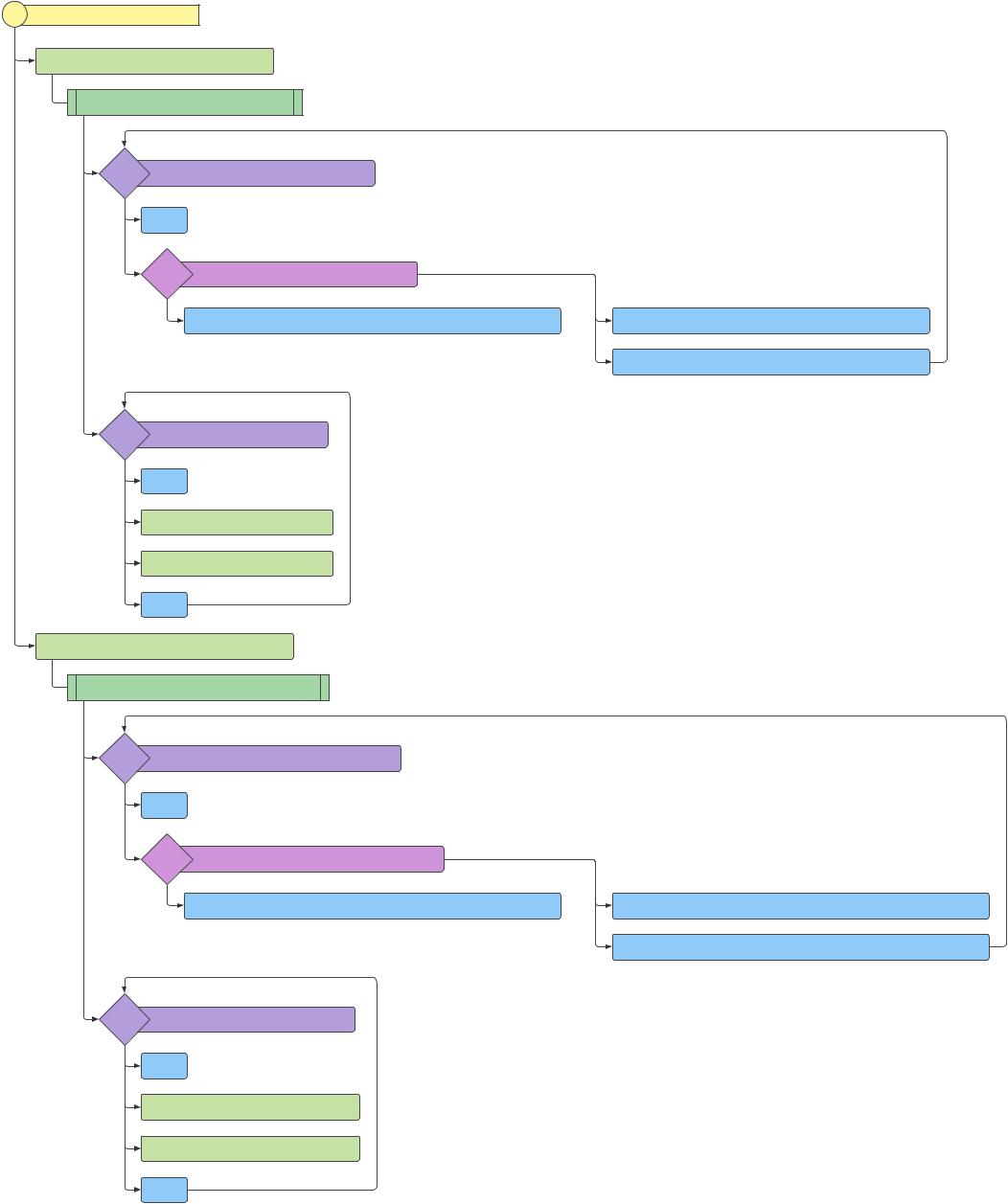
If falls below zero health during the game, they will be disconnected from the game. For example, if a low-health player is hit by a bullet, they will be discon-nected, removed from the game and the other clients will be updated.

|  |  |
| --- | --- |
| Creating a simple networked game — Using Javascript | 16 |

**Controls**



The game relies using up arrowkey or W to move a ship towards the current position of the mouse cursor. The player right-click to shoot a limited amount bullet, however, the player must reload with the Space before firing more bullets.



Program: source module

socket.on('bullets-update', \*())

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| function(server\_bullet\_array) | | |  |  |  |  |
| for | i < server\_bullet\_array.length | |  |  |  |  |
|  | i++ |  |  |  |  |  |
|  | if | (bullet\_array[i] == undefined) | - |  |  |  |
|  |  |  |  |  |
|  | + |  |  |  |  |  |
|  |  | bullet\_array[i] = game.add.sprite(server\_bullet\_ar... | | | bullet\_array[i].x = server\_bullet\_array[i].x |  |
|  |  |  |  |  | bullet\_array[i].y = server\_bullet\_array[i].y |  |
| for | i < bullet\_array.length | |  |  |  |  |
|  | i++ |  |  |  |  |  |
|  | bullet\_array[i].destroy() | |  |  |  |  |
|  | bullet\_array.splice(i, 1) | |  |  |  |  |
|  | i-- |  |  |  |  |  |
| socket.on('healthPack-update', \*()) | | |  |  |  |  |
| function(server\_healthpack\_array) | | |  |  |  |  |
| for | i < server\_healthpack\_array.length | |  |  |  |  |
|  | i++ |  |  |  |  |  |
|  | if | (healthPack\_array[i] == undefined) | | - |  |  |
|  |  |  |  |

+

healthPack\_array[i] = game.add.sprite(server\_healt...

for i < healthPack\_array.length

i++

healthPack\_array[i].destroy()

healthPack\_array.splice(i, 1)

i--

healthPack\_array[i].x = server\_healthpack\_array[i]...

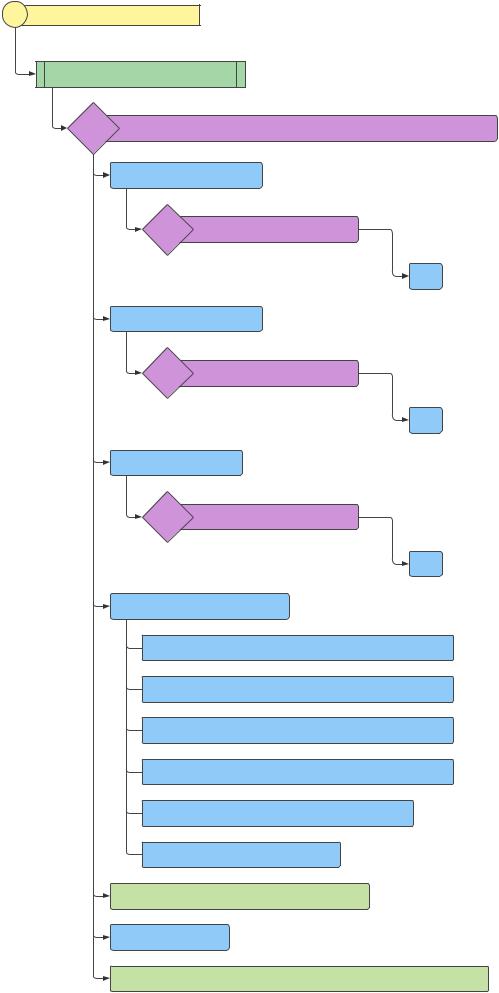
healthPack\_array[i].y = server\_healthpack\_array[i]...

Figure 13: Flow Chart for Updating Hearts and Bullets Client-Side [[3](#page20)]

|  |  |
| --- | --- |
| Creating a simple networked game — Using Javascript | 17 |

**Using Socket.io to Send/Receive Messages**

Using [Socket.io](#page20) involves listening for a update client-side, emitting a message server-side. A er the game objects are created on the game server, an array containing current information is sent to all clients.



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Program: source module | |  |  |  |
| function spawnDriftWood() | | |  |  |
| if | (numDriftWood < maxDriftWood && spawnObjectsAllowe... | | |  |
|  | + |  |  |  |
|  | var plusOrMinusx = | |  |  |
|  | ? | (Math.random() < 0.5) | - |  |
|  |  |  |
|  | + |  |  |  |
|  |  |  | 1 |  |
|  | var plusOrMinusy = | |  |  |
|  | ? | (Math.random() < 0.5) | - |  |
|  |  |  |
|  | + |  |  |  |
|  |  |  | 1 |  |
|  | var rotateDir = | |  |  |
|  | ? | (Math.random() < 0.5) | - |  |
|  |  |  |

+

1

var new\_driftWood = {\*}

x: Math.random() \* 0.75 \* WORLD\_SIZE.w + 50

1. Math.random() \* 0.75 \* WORLD\_SIZE.h + 25 speed\_x: Math.random() \* 0.5 \* plusOrMinusx speed\_y: Math.random() \* 0.5 \* plusOrMinusy rotation: Math.random() \* 2 \* Math.PI rotateDirection: rotateDir

driftWood\_array.push(new\_driftWood)

numDriftWood++

console.log('Spawning DriftWood (' + new\_driftWood...

Figure 14: Logic To Spawn mini-ships Server-Side [[3](#page20)]

The mini-ships spawn randomly in the game world, and move in a constant random x,y direction and rotate at a rate of *π/*200 see Table [4](#page12).

Listing 1: [Socket.io](#page20) client and server commands to listen and receive data

socket . on( 'player - heal ', **function** (id) // listen for a heart co l lision

|  |  |
| --- | --- |
| Creating a simple networked game — Using Javascript | 18 |

io . emit ( 'player - heal ' ,id) ; // Tell the player that their ship collides with a heart

Although the location of players, and other game objects is actively tracked on the server-side, constant updates from the server are sent to the client.

However, for this game, rendering sprites, playing sound and management of game interface is handled client-side.



Figure 15: Player Sprite changing as health decreases

**Conclusion**

Creating a simple networked game is challenging because it requires specialized libraries such as [Socket.io](#page20), periodically updating the server with new information, and debugging both client and server together. Overall, implementing a [Client](#page19) [Server Architecture](#page19) involves understanding how to listen for data client-side and emit data server-side while concious of particular features appropriate for the client and server, respectively.

**List of Terms**

**CSS** Cascading Style Sheets describes how HTML elements will be displayed to the user. [3](#page4)

**Client Server Architecture**

computing model in which the server hosts, delivers and manages most of the resources and services to be consumed by the client. [2](#page3), [3](#page4), [9](#page10), [12](#page13)

**Express.js**

light-weight web application framework used to fetch assets. [6](#page7)

|  |  |
| --- | --- |
| Creating a simple networked game — Using Javascript | 19 |

**HTML**

HyperText MarkUp Language is a standarized language that provides struc-ture to web pages. [3](#page4)

**JavaScript**

scripting language used to implement complex logic in web pages. [3](#page4), [4](#page5)

**Node.js**

open-source cross platform JavaScript library for executing JavaScript code server-side. [6](#page7), [9](#page10), [14](#page15)

**NPM** package management for [Node.js](#page20) that simplifies installation, uses infor-mation from json file to install packages [6](#page7)

**Phaser**

fun open source HTML5 game framework uses canvas rendering build on Pixi.js. [10](#page11), [13](#page14)

**Socket.io**

javascript library that allows communcation bewteen client and servers. It contains a client-side library and a server-side library running on [Node.js](#page20). [2](#page3), [3](#page4), [6](#page7), [12](#page13)

**References**

**Resources Used**

Multiple tutorials and libraries were examined including PubNub (library for networked applications [[8](#page21)] ), an example of a multiplayer game [[9](#page21)] and examples of real-time games implemented in [Phaser](#page20) [[10](#page21)] and [[11](#page21)].

1. *Javascript framework for client-server*. [Online] Available:[https://socket.](https://socket.io/docs/)[io/docs/](https://socket.io/docs/). Javascript. Version 2.0.1. Accessed Dec 10, 2017.
2. Photon Storm. *Phaser Open-Source HTML 5 Library Game Engine*. Javascript. Version 2.6.2. Accessed Dec 10, 2017.
3. *js2flowchart - a visualization library*. [Online] Available:[https://github.](https://github.com/Bogdan-Lyashenko/js-code-to-svg-flowchart)[com / Bogdan - Lyashenko / js - code - to - svg - flowchart](https://github.com/Bogdan-Lyashenko/js-code-to-svg-flowchart). Version 1.0.4. Accessed Dec 10, 2017.

|  |  |
| --- | --- |
| Creating a simple networked game — Using Javascript | 20 |

1. Google. *Chrome DevTools*. [Online] Available: [https://developers.google.](https://developers.google.com/web/tools/chrome-devtools/) [com/web/tools/chrome-devtools/](https://developers.google.com/web/tools/chrome-devtools/). Version 62.0.3202.94 (O icial Build)

(64-bit). Accessed Dec 10, 2017.

1. Mozilla Firefox. *Firefox web browser*. Version 56.0.2 (64-bit).
2. Microso . *Microso Edge Browser*. [Online] Available: [https://www.microsoft.](https://www.microsoft.com/en-ca/windows/microsoft-edge) [com/en-ca/windows/microsoft-edge](https://www.microsoft.com/en-ca/windows/microsoft-edge). Version 40.15063.674.0.
3. w3schools. *CSS Legal Color Values*. [Online] Available: [https : / / www](https://www.w3schools.com/cssref/css_colors_legal.asp) . [w3schools . com / cssref / css \_ colors \_ legal . asp](https://www.w3schools.com/cssref/css_colors_legal.asp). Accessed Dec 10, 2017.
4. *Platform Game Using Phasor*. [Online] Available:[https://github.com/](https://github.com/pubnub-dsn/Ninja-Multiplayer-Platformer)[pubnub-dsn/Ninja-Multiplayer-Platformer](https://github.com/pubnub-dsn/Ninja-Multiplayer-Platformer). Accessed Dec 10, 2017.
5. *Getting started with Phaser with web server*. [Online] Available:[http://](http://shawnhymel.com/1148/getting-started-with-phaser-part-1-web-server/)[shawnhymel.com/1148/getting-started-with-phaser-part-1-web-server/](http://shawnhymel.com/1148/getting-started-with-phaser-part-1-web-server/). Accessed Dec 10, 2017.
6. *Tic Tac Toe Example Using Phasor*. [Online] Available:[https://pubnub.](https://pubnub.github.io/api-guide-with-tictactoe/plain.html?id=1773)[github . io / api - guide - with - tictactoe / plain . html ? id = 1773](https://pubnub.github.io/api-guide-with-tictactoe/plain.html?id=1773). Ac-cessed Dec 10, 2017.
7. *Creating a multiplayer game using websockets and node*. [Online] Available:<http://rawkes.com/articles/creating-a-real-time-multiplayer-game-with-websockets-and-node.html>. Accessed Dec 10, 2017.

**Complete Code for Multiplayer Game**

The game can be installed locally by changing to the main directory containing the game using [NPM](#page20) to install the dependencies listed in **package.json**, and going to a web browser. The following commands are run to install the game:

npm install

node server

Go to http://localhost:8000

Listing 2: Dependencies for game listed in package.json

* {

|  |  |
| --- | --- |
| 2 | " name " : "David 's Pirate Game " , |

* " version " : " 1 .1. 1",

|  |  |
| --- | --- |
| Creating a simple networked game — Using Javascript | 21 |

* " d e s c r i p t i o n " : " Simple N e tworked Game --- an simple

game that i m p l e me n t s a client - server a r c h i t e c t u r e ."

,

* " main " : " server . js" ,

6" scripts " : {

7" start " : " node server . js"

* },
* " d e p e n d e n c i e s ": {

1. " express ": "^ 4 .15 .2" ,
2. " socket . io" : "^ 2. 0.1 "
3. },
4. " engines " : {
5. " node ": "6 . 10 . x"
6. },
7. " r e p o s it o r y " : {
8. " url ": " https : // github . com/ F r i e n d l y U s e r / Pirates - Game "
9. },
10. " license " : " MIT " ,
11. " keywords " : [
12. " node ",
13. " phaser " ,
14. " express "
15. ]
16. }

Listing 3: HTML Code for the Pirates Game

**<html >**

**<head >**

**<meta** charset =" UTF-8 " **/>**

**<title >** Ship Battle M u l t i p l a y e r Game **</ title >**

* ! -- Load the Phaser game library -- **>**

**<script** src =" https: **//** cdnjs . c l o u d f l a r e . com **/** ajax **/** libs **/** phaser **/**

2.6.2 **/** phaser . min . js" **></ script >**

**<script** src="**/** socket . io **/** socket . io . js" **></ script > <**! -- Some simple styles and fonts -- **>**

* link rel = " s t y l e s he e t " type = " text **/** css " href = " css **/** game .

css " **>**

|  |  |
| --- | --- |
| Creating a simple networked game — Using Javascript | 22 |

* link rel= " s t y l e sh e e t " type = " text **/** css " href = " css **/** s e r v e r M e s s a g e s . css " **>**

**</ head >**

**<body >**

**<script** type =" text **/** j a v a s cr i p t " src = "js **/** h e a l t h M e t e r . js" **> </**

**script >**

**<script** type =" text **/** j a v a s cr i p t " src = "js **/** client . js" **> </**

**script >**

* div id= " p h a s e r - e x a m p l e " **></** div **>**
* ! -- **<** ul id=" messages " **></** ul **>**
  + form action ="" **>**

**<** input id="m"a u t o c o m p l e t e ="off " **/><** button **>** Send **</** button **>**

**</** form **>** -- **>**

**<**! -- **<** div id="m" **></** div **>** -- **>**

* ! -- **<script** src= " https: **//** code . jquery . com **/** jquery-1 .11.1. js" **></** **script >** -- **>**
  + ! -- chatbox logic goes here
  + ul id=" messages " **></** ul **>**
  + form action ="" **>**

**<** input id="m"a u t o c o m p l e t e ="off " **/><** button **>** Send **</** button **> </** form **>**

* **>**
* footer **>**
* ul id=" log" **>**
* pre **>**

**<**! -- **<script** type =" text **/** j a v a s c ri p t " src = "js **/** l o g g i n g O n S c r e e n

* js" **> </ script >** -- **>** **</** pre **>**

**</** ul **>**

**</** footer **> </ body >**

**</ html >**

Listing 4: Client-side logic

**var** AS S ET\_URL = " assets /"

* We first i n i t i a l i z e the phaser game object **var** W I N D O W \_ W I D T H = 750;

**var** W I N D O W \_ H E I G H T = 500; **var** H E A L T H\_ M A X = 100;

**var** game = **new** Phaser . Game ( WINDOW\_WIDTH , WINDOW\_HEIGHT , Phaser .AUTO , 'phaser - example ', { preload : preload , create : create , update : GameLoop } ) ;

**var** W O R L D \_S I Z E = {w :1500 , h :1000}; **var** n u m o f E n e m y P l a y e r s = 0;

**var** w a t e r \_ t i l e s = [];

|  |  |
| --- | --- |
| Creating a simple networked game — Using Javascript | 23 |

* arrays of objects that the player will interact with **var** b u l l e t \_ a r r a y = [];

**var** h e a l t h P a c k \_ a r r a y = []; **var** d r i f t W o o d \_ a r r a y = [];

**var** sh o tsLeft ;// text field c o n t a i ni n g number of c a n n o n b a l l sthe player has r e maining before r e loading

**var** n u m O f E n e m i e s T e x t ;

**var** p l a y e r s C o n n e c t i n g T e x t ; **var** loadText ;

**var** socket ;// Declare it in this scope , i n i t i a l i z e in the `create ` function

**var** o t h e r \_ p l a y e r s = {};

**var** b u l l e t S p e e d = 17.5;// speed of the bullet **var** player = {

sprite : **null** , // Will hold the sprite when it 's created speed\_x :0 , // This is the speed it 's c u rrently moving at

speed\_y :0 ,

speed :0.5 , // This is the p a rameter for how fast it should move

friction :0.95 ,

health :100 ,

bullets :15 ,

* shipType :null , alive : **true** , shot : **false** , update : **function** () {

// Lerp rotation towards mouse

**var** dx = ( game . input . m o u s e P o i n t e r . x + game . camera . x) - **this** .

sprite . x;

**var** dy = ( game . input . m o u s e P o i n t e r . y + game . camera . y) - **this** .

sprite . y;

**var** angle = Math . atan2 (dy ,dx) - Math . PI /2;

**var** dir = ( angle - **this** . sprite . rotation ) / ( Math . PI \* 2) ;

dir -= Math . round (dir ) ;

dir = dir \* Math . PI \* 2;

**this** . sprite . rotation += dir \* 0.1;

// Move forward

**if** ( game . input . keyboard . isDown ( Phaser . Keyboard . W) || game .

input . keyboard . isDown ( Phaser . Keyboard . UP) ) {

* used to decrease speed of player as health d e creases **var** h e a l t h Le f t = **this** . health - H E A L T H\_ M A X ;

**this** . speed\_x += Math . cos ( **this** . sprite . rotation + Math . PI /2)

\* **this** . speed \* **this** . health /100;

**this** . speed\_y += Math . sin ( **this** . sprite . rotation + Math . PI /2)

|  |  |
| --- | --- |
| Creating a simple networked game — Using Javascript | 24 |

* **this** . speed \* **this** . health /100;

}

**this** . sprite .x += **this** . speed\_x ; **this** . sprite .y += **this** . speed\_y ;

// change player sprite based on health .

**this** . sprite = c h a n g e S p r i t e B a s e d O n H e a l t h ( **this** . sprite , **this** .

health )

**this** . speed\_x \*= **this** . friction ;

**this** . speed\_y \*= **this** . friction ;

* inform the player that they can reload
* Shoot bullet

**if** ( game . input . a c t i v e P o i n t e r . l e f t B u t to n . isDown && ! **this** . shot )

{

**if** ( **this** . bullets <= 0) {

w r i t e T o H T M L L o g ( 'Press spacebar or enter to reload bullets . ') ;

lo a dSound . play () ;

**this** . shot = **true** ;

}

**else**

{

**var** speed\_x = Math . cos ( **this** . sprite . rotation + Math . PI /2)

\* b u l l e t S p e e d ;

**var** speed\_y = Math . sin ( **this** . sprite . rotation + Math . PI /2)

\* b u l l e t S p e e d ;

**this** . bullets = **this** . bullets -1 ;

u p d a t e b u l l e t s ( **this** . bullets ) ;

**this** . shot = **true** ;

// Tell the server we shot a bullet

socket . emit ( 'shoot - bullet ' ,{x: **this** . sprite .x,y: **this** . sprite .y, angle : **this** . sprite . rotation , speed\_x : speed\_x , speed\_y : speed\_y })

fi r eSound . play () ;

}

}

**else if** (( game . input . keyboard . isDown ( Phaser . Keyboard .SPACEBAR ) || game . input . keyboard . isDown ( Phaser . Keyboard . ENTER ) ) && **this** . bullets <= 0) {

**this** . bullets = **this** . bullets +5;

u p d a t e b u l l e t s ( **this** . bullets ) ;

w r i t e T o H T M L L o g ( " bullets : (" + **this** . bullets + ') \t \t' + "

Player Health : ("+ **this** . health + ')')

**var**

**;**

|  |  |
| --- | --- |
| Creating a simple networked game — Using Javascript | 25 |

**this** . shot = **true** ;

}

**if** (! game . input . a c t i v e P o i n t e r . l e f t B u t t o n . isDown ) **this** . shot = **false** ;

u p d a t e H e a l t h B a r ( HEALTH\_MAX - player . health )

* To make player flash when they are hit , set player . spite . alpha = 0

**if** ( **this** . sprite . alpha < 1) {

**this** . sprite . alpha += (1 - **this** . sprite . alpha ) \* 0.16;

} **else** {

**this** . sprite . alpha = 1;

}

// Tell the server we 've moved

socket . emit ( 'move - player ' ,{x: **this** . sprite .x,y: **this** . sprite .y,

angle : **this** . sprite . rotation , health : **this** . health })

}

};

**function** C r e a t eS h i p (type ,x,y, angle ) {

* type is an int that can be between 1 and 6 i n clusive
* returns the sprite just created

sprite = game . add . sprite (x,y, 'ship ' + String ( type ) + ' \_1 ')

game . physics . enable ([ sprite ] , Phaser . Physics . ARCADE ) ;

sprite . rotation = angle ;

sprite . anchor . setTo (0.5 ,0.5) ;

sprite . body . c o l l i d e W o r l d B o u n d s = **true** ;

sprite . body . bounce . setTo (1 , 1) ;

**return** sprite ;

}

**function** preload () {

game . load . c r o s s O r i g i n = " An o nymous ";

game . stage . b a c k g r o u n d C o l o r = " #3399 DA" ;

game . load . o n F i l e C o m p l e t e . add ( fileComplete , **this** ) ;

* Load all the ships for ( **var** i=1;i <=6; i ++) {

game . load . image ( 'ship '+ String (i) + ' \_1 ' , A S SET\_URL + 'ship '+ String (i) + ' \_1 . png ') ;

|  |  |
| --- | --- |
| Creating a simple networked game — Using Javascript | 26 |

game . load . image ( 'ship '+ String (i) + ' \_2 ' , A S SET\_URL + 'ship '+ String (i) + ' \_2 . png ') ;

game . load . image ( 'ship '+ String (i) + ' \_3 ' , A S SET\_URL + 'ship '+ String (i) + ' \_3 . png ') ;

game . load . image ( 'ship '+ String (i) + ' \_4 ' , A S SET\_URL + 'ship '+ String (i) + ' \_4 . png ') ;

}

game . load . image ( 'bullet ' , A S SET\_URL + ' c a n n o n \_ b a l l . png ') ;

game . load . image ( 'water ' , A S SET\_URL + 'w a t e r \_t i l e . png ') ;

game . load . image ( 'h e a l t h Pa c k ' , A S SET\_URL + 'heart . png ') ;

game . load . image ( 'dr i ftWood ' , A S SET\_URL + ' d i n g h y L a r g e 3 . png ')

// play ready sound once game is loaded

game . load . audio ( 'loaded ' , A S SET\_URL + 'sound /' + 'ready . wav ') ; game . load . audio ( 'hit ', A S SET\_URL + 'sound /' + 'thud1 . wav ') ; game . load . audio ( ' f i r e C a n n o n b a l l ' , A S SET\_URL + 'sound /' + '

c a n n o n ba l l . wav ') ;

game . load . audio ( 's o u n d t ra c k ' , A S SET\_URL + 'sound /' + 'pirates .

wav ') ;

game . load . audio ( 'healing ' , A S SET\_URL + 'sound /' + 'healing . wav

* ) ;

loadText = game . add . text ( game . world . centerX , game . world . centerY , ' Co n nected to Server ,' + '\n' + 'Waiting for the game to load ') ;

// text = game . add . text ( , " - phaser gradient text -");

* Centers the text loadText . anchor . set (0.5) ; loadText . align = 'center ';
* Our font + size loadText . font = 'Arial '; loadText . f o n t W e i gh t = 'bold '; loadText . fontSize = 35;
* Here we create a linear gradient on the Text context .
* This uses the exact same method of creating a gradient as you do on a normal Canvas context .

**var** grd = loadText . context . c r e a t e L i n e a r G r a d i e n t (0 , 0 , 0 ,loadText . height ) ;

* Add in 2 color stops

grd . a d d C o l o r S t o p (0 , '#00008 B') ;

grd . a d d C o l o r S t o p (1 , '#006400 ') ;

// And apply to the Text

|  |  |
| --- | --- |
| Creating a simple networked game — Using Javascript | 27 |

loadText . fill = grd ;

game . load . o n L o a d C o m p l e t e . add ( loadComplete , **this** ) ;

}

* This callback is sent the f o llowing p a r a m e t e r s :

**function** f i l e C o m p l e t e ( progress , cacheKey , success , totalLoaded , t o t a l F il e s ) {

loadText . setText ( ' Co n nected to Server ,' + '\n' + 'Waiting for the game to load . \n' + " File Complete : " + progress + "% -

* + t o t a l L o a d e d + " out of " + t o t a l F i l es ) ;

// text . setText (" File Complete : " + progress + "% - " + t o t a l L o a d e d + " out of " + t o t a l F i l e s );

}

**function** l o a d C o m p l e t e () {

// console . log (" Load Complete ");

lo a dSound = game . add . audio ( 'loaded ') ;

lo a dSound . play () ;

loadText . setText ( '') ;

w r i t e T o H T M L L o g ( 'Player Ship Created and ready to go . ') ;

w r i t e T o H T M L L o g ( 'Controls : Right Click to Shoot and use W or UP key to move . ') ;

w r i t e T o H T M L L o g ( 'Controls : The ship will move towards the mouse cursor . ') ;

}

**function** create () {

// turn on game physics

game . physics . s t a r t S y s t e m ( Phaser . Physics . ARCADE ) ;

* play music see https :// phaser . io/ examples /v2/ audio / loop music = game . add . audio ( 's o u n d tr a c k ') ;

hitSound = game . add . audio ( 'hit ') ;

he a lSound = game . add . audio ( 'healing ') ; music . loopFull (0.25) ;

* music . volume = 0.25;
* music . onLoop . add( this );

fi r eSound = game . add . audio ( ' f i r e C a n n o n b a l l ') ; // Create water tiles

for ( **var** i=0;i <= W O R L D \_S I Z E . w /64+1; i ++) { for ( **var** j=0;j <= W O R L D \_S I Z E . h /64+1; j ++) {

**var** t i l e \_ s p r i t e = game . add . sprite (i \* 64 , j \* 64 , 'water ')

;

t i l e \_ s p r i t e . anchor . setTo (0.5 ,0.5) ;

t i l e \_ s p r i t e . alpha = 0.5;

|  |  |
| --- | --- |
| Creating a simple networked game — Using Javascript | 28 |

w a t e r \_ t i l e s . push ( t i l e \_ s p r i t e ) ;

}

}

game . stage . d i s a b l e V i s i b i l i t y C h a n g e = **true** ; game . world . se t Bounds (0 , 1500 , 0 , 1200) ; c r e a t e H e a l t h B a r ()

* Create player
* var p l a y e r \_ s h i p \_ t y p e = String (1) ;
* Range of Math . random is [0 ,1) , not i n clusive

**var** p l a y e r \_ s h i p \_ t y p e = String ( Math . floor (6\* Math . random () ) +1) ;// randomly spawn player sprite

player . sprite = game . add . sprite ( Math . random () \* W O R L D \_ S I Z E .w/2

* W O R L D \_S I Z E .w/2 , Math . random () \* W O R L D \_ S I ZE . h/2 + W O R L D \_ S I Z E

. h/2 , 'ship '+ p l a y e r \_ s h i p \_ t y p e + '\_1 ') ;

|  |  |  |
| --- | --- | --- |
| game . physics . enable ([ player ] , | Phaser . Physics . ARCADE ) ; | |
| player . sprite . anchor . setTo (0.5 | ,0.5) | ; |
| sh o tsLeft = game . add . text (625 , | 475 , | " Shots Left : " + player . |

bullets , { font : " bold 16 px Arial " , fill : "# ffffff " , align : "

center " }) ;

sh o tsLeft . f i x e d T o C a m e r a = **true** ;

n u m O f E n e m i e s T e x t = game . add . text (5 , 475 , " Player ID: " , { font

* + " bold 16 px Arial " , fill : "# ffffff " , align : " center " }) ; n u m O f E n e m i e s T e x t . f i x e d T o C a m e r a = **true** ;

game . world . se t Bounds (0 , 0 , W O R L D \_ SI Z E .w, W O R L D \_ S I ZE .h) ;

* if ( n u m o f E n e m y P l a y e r s <= 0 && p l a y e r s C o n n e c t i n g T e x t == ' ') { **if** ( n u m o f E n e m y P l a y e r s <= 0) {

w r i t e T o H T M L L o g ( 'Waiting for players to join ') ;

p l a y e r s C o n n e c t i n g T e x t = game . add . text (250 ,10 , " Waiting for other players to join " , { font : " bold 16 px Arial " , fill : "# ffffff " , align : " center " }) ;

p l a y e r s C o n n e c t i n g T e x t . f i x e d T o C a m e r a = **true** ; // console . log( p l a y e r s C o n n e c t i n g T e x t )

}

game . camera . x = player . sprite .x - W I N D O W \_ W I D T H /2; game . camera . y = player . sprite .y - W I N D O W \_ H E I G H T /2;

socket = io () ; // This triggers the 'c o n n e c ti o n ' event on the server

socket . emit ( 'new - player ' ,{x: player . sprite .x,y: player . sprite .y, angle : player . sprite . rotation , type : player\_ship\_type , health : player . health })

// Listen for other players c o n n e c t i n g

socket . on( 'update - players ', **function** ( p l a y e r s \_ d a t a ) {

|  |  |
| --- | --- |
| Creating a simple networked game — Using Javascript | 29 |

**var** p l a y e r s \_ f o u n d = {};

* Loop over all the player data received for ( **var** id **in** p l a y e r s \_ d a t a ) {

|  |  |  |  |
| --- | --- | --- | --- |
| // | If the player | hasn 't | been created yet |
| **if** ( o t h e r \_ p l a y e r s [id] == | | | u n defined && id != socket . id) { // |
| Make | sure you don 't | create | yourself |

**var** data = p l a y e r s \_ d a t a [id ];

**var** p = C r e a t eS h i p ( data . type , data .x, data .y, data . angle ) ;p. health = data . health ; // set up the health ? // // console . log(p . health )

o t h e r \_ p l a y e r s [id] = p;

// console . log (" Created new player at (" + data . x + " , "

+ data . y + ") ");

w r i t e T o H T M L L o g ( 'Player [' + id + ']' + ' Joined the Game

. ') ;

**if** ( p l a y e r s C o n n e c t i n g T e x t . text =='') {

p l a y e r s C o n n e c t i n g T e x t . setText ( 'Player Joined ') ;

p l a y e r s C o n n e c t i n g T e x t . alpha = 1;

// console . log (' Messages goes here ?')

}

n u m o f E n e m y P l a y e r s ++;

}

p l a y e r s \_ f o u n d [id] =

**true** ;

* Update p o sition s of other players **if** (id != socket . id) {

o t h e r \_ p l a y e r s [id ]. target\_x = p l a y e r s \_ d a t a [id ]. x; //

Update target , not actual position , so we can i n t e r p o l a t e

o t h e r \_ p l a y e r s [id ]. target\_y = p l a y e r s \_ d a t a [id ]. y;

o t h e r \_ p l a y e r s [id ]. t a r g e t \_ r o t a t i o n = p l a y e r s \_ d a t a [id ].

angle ;

o t h e r \_ p l a y e r s [id ]. type = p l a y e r s \_ d a t a [id ]. type ;

o t h e r \_ p l a y e r s [id ]. health = p l a y e r s \_ d a t a [id ]. health ;

// // console . log( o t h e r \_ p l a y e r s [id ])

* change player sprite based on health .
* move this to other function a f t e r w a r d s .
* console . log ( o t h e r \_ p l a y e r s [id ]. health )

o t h e r \_ p l a y e r s [id] = c h a n g e S p r i t e B a s e d O n H e a l t h ( o t h e r \_ p l a y e r s [id] , o t h e r \_ p l a y e r s [id ]. health )

}

}

* Check if a player is missing and delete them for ( **var** id **in** o t h e r \_ p l a y e r s ) {

|  |  |
| --- | --- |
| Creating a simple networked game — Using Javascript | 30 |
| **if** (! p l a y e r s \_ f o u n d [id ]) { |  |
| n u m o f E n e m y P l a y e r s - -; |  |
| w r i t e T o H T M L L o g ( 'Player [' + id + ']' + ' D i s c o n n e c t e d | |
| from the Game . ') ; |  |
| **if** ( p l a y e r s C o n n e c t i n g T e x t . text =='') | { |
| p l a y e r s C o n n e c t i n g T e x t . setText ( 'Player D e stroyed ') ; | |
| p l a y e r s C o n n e c t i n g T e x t . alpha = 1; |  |
| } |  |
| o t h e r \_ p l a y e r s [id ]. destroy () ; |  |
| delete o t h e r \_ p l a y e r s [id ]; |  |

}

}

// print list of players in game screen

n u m O f E n e m i e s T e x t . setText (" Enemy Ships : " + n u m o f E n e m y P l a y e r s ) ;

// print waiting for players to connect message

**if** ( n u m o f E n e m y P l a y e r s > 0 & p l a y e r s C o n n e c t i n g T e x t . text != ''

* p l a y e r s C o n n e c t i n g T e x t . alpha != 0) { game . add . tween ( p l a y e r s C o n n e c t i n g T e x t )
  + to ({ alpha : 0} , 1000 , Phaser . Easing . Default , **true** , 3000) . o n C o m p le t e . add( **function** () {

**if** ( p l a y e r s C o n n e c t i n g T e x t . text !='') {

* console . log(' Other Players have joined ');
* console . log (" This is called when the tween is

done .") ;

p l a y e r s C o n n e c t i n g T e x t . setText ( '') ; // destroy game text ? consider after adding c o llision objects .

* game . world . remove ( p l a y e r s C o n n e c t i n g T e x t );

}

} , **this**

) ;

}

// helpful awaiting for players to connect

})

// Listen for bullet update events

socket . on( 'bullets - update ', **function** ( s e r v e r \_ b u l l e t \_ a r r a y ) {

// If there 's not enough bullets on the client , create them for ( **var** i=0;i< s e r v e r \_ b u l l e t \_ a r r a y . length ;i ++) {

**if** ( b u l l e t \_ a r r a y [i] == u n defined ) {

b u l l e t \_ a r r a y [i] = game . add . sprite ( s e r v e r \_ b u l l e t \_ a r r a y [i ]. x, s e r v e r \_ b u l l e t \_ a r r a y [i]. y, 'bullet ') ;

} **else** {

|  |  |  |  |
| --- | --- | --- | --- |
| Creating a simple networked game — Using Javascript | | | 31 |
| // Otherwise , just | update it! | |  |
| b u l l e t \_ a r r a y [i]. x | = | s e r v e r \_ b u l l e t \_ a r r a y [i]. x; |  |
| b u l l e t \_ a r r a y [i]. y | = | s e r v e r \_ b u l l e t \_ a r r a y [i]. y; |  |
| } |  |  |  |

}

// O t herwise if there 's too many , delete the extra

for ( **var** i= s e r v e r \_ b u l l e t \_ a r r a y . length ;i< b u l l e t \_ a r r a y . length ;i ++) {

b u l l e t \_ a r r a y [i]. destroy () ;

b u l l e t \_ a r r a y . splice (i ,1) ;

i - -;

}

})

// Listen for h e a l t h P a c k update events

socket . on( 'healthPack - update ' , **function** ( s e r v e r \_ h e a l t h p a c k \_ a r r a y ) {

// If there 's not enough health packs on the client , create them

// could do the moving client side , but hmm

for ( **var** i=0;i< s e r v e r \_ h e a l t h p a c k \_ a r r a y . length ;i ++) {

// add d r iftwood to game if it doesn 't already exist

**if** ( h e a l t h P a c k \_ a r r a y [i] == u n defined ) {

h e a l t h P a c k \_ a r r a y [i] = game . add . sprite (

s e r v e r \_ h e a l t h p a c k \_ a r r a y [i]. x, s e r v e r \_ h e a l t h p a c k \_ a r r a y [i]. y, ' h e a l t h Pa c k ') ;

* + // console . log( h e a l t h P a c k \_ a r r a y [i]) } **else** {
  + Otherwise , just update it!

h e a l t h P a c k \_ a r r a y [i]. x = s e r v e r \_ h e a l t h p a c k \_ a r r a y [i]. x; h e a l t h P a c k \_ a r r a y [i]. y = s e r v e r \_ h e a l t h p a c k \_ a r r a y [i]. y;

// // console . log( h e a l t h P a c k \_ a r r a y [i])

}

}

// O t herwise if there 's too many , delete the extra

for ( **var** i= s e r v e r \_ h e a l t h p a c k \_ a r r a y . length ;i< h e a l t h P a c k \_ a r r a y .

length ;i ++) {

h e a l t h P a c k \_ a r r a y [i]. destroy () ;

h e a l t h P a c k \_ a r r a y . splice (i ,1) ;

i - -;

}

|  |  |
| --- | --- |
| Creating a simple networked game — Using Javascript | 32 |

})

// Listen for d r iftWood update events

socket . on( 'driftWood - update ' , **function** ( s e r v e r \_ d r i f t W o o d \_ a r r a y ) { for ( **var** i=0;i< s e r v e r \_ d r i f t W o o d \_ a r r a y . length ;i ++) {

// add d r iftWood to game if it doesn 't already exist

**if** ( d r i f t W o o d \_ a r r a y [i] == u n defined ) {

d r i f t W o o d \_ a r r a y [i] = game . add . sprite (

s e r v e r \_ d r i f t W o o d \_ a r r a y [i].x, s e r v e r \_ d r i f t W o o d \_ a r r a y [i].y, ' dr i ftWood ') ;

d r i f t W o o d \_ a r r a y [i]. rotation = s e r v e r \_ d r i f t W o o d \_ a r r a y [i].

rotation ;

} **else** {

// Otherwise , just update it!

d r i f t W o o d \_ a r r a y [i]. x = s e r v e r \_ d r i f t W o o d \_ a r r a y [i]. x;

d r i f t W o o d \_ a r r a y [i]. y = s e r v e r \_ d r i f t W o o d \_ a r r a y [i]. y;

d r i f t W o o d \_ a r r a y [i]. rotation = s e r v e r \_ d r i f t W o o d \_ a r r a y [i].

rotation ;

// // console . log( h e a l t h P a c k \_ a r r a y [i])

}

**var** dx = d r i f t W o o d \_ a r r a y [i]. x - player . sprite .x;

**var** dy = d r i f t W o o d \_ a r r a y [i]. y - player . sprite .y;

**var** dist = Math . sqrt (dx \* dx + dy \* dy) ;

* // console . log ( dist );
* If the players get close enough to collide **if** ( dist < 45) {

// check if the player collided with d r iftwood

game . physics . arcade . collide ( d r i f t W o o d \_ a r r a y [i] , player .

sprite ) ;

* console . log ('Big c o llision happened :');
* check if players are c o lliding

h i t D r i f t W o o d = c h e c k O v e r l a p ( d r i f t W o o d \_ a r r a y [i] , player .

sprite )

**if** ( h i t D r i f t W o o d ) {

hitSound . volume = 0.5;

hitSound . play () ;

player . health = player . health -1;

* push player back far enough to avoid c o llision player . sprite .x -= dx \*0.15;

player . sprite .y -= dy \*0.15;

* player . update ();
* socket . emit ('move - player ',{x: player . sprite .x,y: player . sprite .y, angle : player . sprite . rotation , health : player . health })

}

|  |  |
| --- | --- |
| Creating a simple networked game — Using Javascript | 33 |

}

}

// O t herwise if there 's too many , delete the extra

for ( **var** i= s e r v e r \_ d r i f t W o o d \_ a r r a y . length ;i< d r i f t W o o d \_ a r r a y .

length ;i ++) {

d r i f t W o o d \_ a r r a y [i]. destroy () ;

d r i f t W o o d \_ a r r a y . splice (i ,1) ;

i - -;

}

})

* Listen for any player hit events and make that player flash socket . on( 'player - hit ', **function** (id) {

**if** (id == socket . id) {

// If this is you

player . sprite . alpha = 0;

player . health = player . health - 1; // why not use a fraction lol .

hitSound . volume = 0.5;

hitSound . play () ;

w r i t e T o H T M L L o g ( 'Ouch , hit by cannon ball . Health : ' + player . health ) ;

* + send i n f o r m a t i o n c o n t a i n i n g the player health so that the sprites updates .

socket . emit ( 'move - player ' ,{x: player . sprite .x,y: player . sprite .y, angle : player . sprite . rotation , health : player . health }) ;

**if** ( player . health <= 0) {socket . emit ( 'd i s c o nn e c t ' ,{}) ; socket . d i s c o n n e c t () ;

* + - console . log (' Game Over : player is d i s c o n n e c t e d from the server ');

w r i t e T o H T M L L o g ( 'Game Over : player is d i s c o n n e c t e d from the server ')

delete player ;

}

} **else** {

* + Find the right player

o t h e r \_ p l a y e r s [id ]. alpha = 0;

}

})

socket . on( 'player - heal ' , **function** (id) {

**if** (id == socket . id) {

* console . log(' You healed :' + id) he a lSound . volume = 0.5;

he a lSound . play () ;

|  |  |
| --- | --- |
| Creating a simple networked game — Using Javascript | 34 |
| **if** ( player . health + 10 >= H E A L T H \_ MA X ) { |  |
| player . health = H E A L T H \_ M A X ; |  |
| } |  |
| **else** { |  |
| player . health += 10; |  |
| w r i t e T o H T M L L o g ( 'You healed because of the | heart : Health ( |
| ' + player . health + ')') ; |  |
| } |  |
| } |  |
| **else** { |  |

* Find the right player
* console . log(' Other player got the heart : ' + id)

}

})

}

**function** GameLoop () {

player . update () ;

// check if the player has been d e stroyed

**if** ( player . health <= 0 && player . alive == **true** ) {

w r i t e T o H T M L L o g ( 'Game Over : D i s c o n n e c t i n g from game server . ')

player . alive = **false** ; // the player is now dead

player . speed\_x = 0;

player . speed\_y = 0;

socket . emit ( 'd i s c o n ne c t ' ,{}) ;

socket . d i s c o n n e ct () ;

// var style = { font : " bold 32 px Arial " , fill : "# fff " ,

b o u n d s A l i g n H : " center " , b o u n d s A l i g n V : " middle " };

* + The Text is p o s i t i o n e d at 0 , 100
  + text = game . add . text ( player . sprite .x, player . sprite . y+50 ,
* Game Over " , style );
  + text . f i x e d T o C a m e r a = true ;

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **var** | t = game . add . text (250 , | | 250 , | " Game | Over " , { font : " bold |
| 32 px | Arial " , fill : "# ffffff " , align : " center " }) ; | | | | |
| t . f i x e d T o C a m e r a = | | **true** ; |  |  |  |
| **var** | t2 = game . add | . text (250 , | 400 , | "No | more bullets " , { font : |

* bold 32 px Arial " , fill : "# ffffff " , align : " center " }) ; t2 . f i x e d T o C a m e r a = **true** ;

//t. c a m e r a O f f s e t . setTo ( player .x, player .y);

}

// Move camera with player

**var** camera\_x = player . sprite . x - W I N D O W \_ W I D T H /2;

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Creating a simple networked game — Using Javascript | | | |  | 35 |
| **var** camera\_y = player . sprite . y - W I N D O W \_ H E I G H T /2; | | | | | |
| game . camera . x += | ( camera\_x | - | game . camera . x) | \* 0.08; | |
| game . camera . y += | ( camera\_y | - | game . camera . y) | \* | 0.08; |

* Each player is r e s p o n s i b l e for bringing their alpha back up on their own client
* Make sure other players flash back to alpha = 1 when they ' re hit

for ( **var** id **in** o t h e r \_ p l a y e r s ) {

**if** ( o t h e r \_ p l a y e r s [id ]. alpha < 1) {

o t h e r \_ p l a y e r s [id ]. alpha += (1 - o t h e r \_ p l a y e r s [id ]. alpha ) \*

0.16;

} **else** {

o t h e r \_ p l a y e r s [id ]. alpha = 1;

}

}

* I n t e r p o l a t e all players to where they should be for ( **var** id **in** o t h e r \_ p l a y e r s ) {

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **var** p | = | o t h e r \_ p l a y e r s [id ]; | | |  |  |
| **if** (p. target\_x != u n defined ) { | | | | |  |  |
| p .x | += | (p . target\_x | - | p.x) | \* | 0.16; |
| p .y | += | (p . target\_y | - | p.y) | \* | 0.16; |

* I n t e p o l a t e angle while avoiding the positive / negative

issue

**var** angle = p . t a r g e t \_ r o t a t i o n ;

**var** dir = ( angle - p . rotation ) / ( Math . PI \* 2) ;dir -= Math . round ( dir ) ;

dir = dir \* Math . PI \* 2;

p . rotation += dir \* 0.16;

}

* check for c o l l i s i o n s between players **if** ( n u m o f E n e m y P l a y e r s > 0) {

**var** dx = p . x - player . sprite .x; **var** dy = p . y - player . sprite .y;

**var** dist = Math . sqrt (dx \* dx + dy \* dy) ;

* + // console . log ( dist );
  + If the players get close enough to collide **if** ( dist < 80) {

game . physics . arcade . collide (p, player . sprite ) ;

* + - console . log ('Big c o llision happened :');
    - check if players are c o lliding

p l a y e r s C o l l i d e = c h e c k O v e r l a p (p, player . sprite )

**if** ( p l a y e r s C o l l i d e ) {

// console . log (' Playing C o llision Sound ')

|  |  |
| --- | --- |
| Creating a simple networked game — Using Javascript | 36 |

hitSound . volume = 0.5;

hitSound . play () ;

player . update () ;

* push player back far enough to avoid c o llision player . sprite .x -= dx \*0.45;

player . sprite .y -= dy \*0.45;

}

**else** {

// console . log ('No C o llision ');

}

}

* io . emit ('player - hit ',id); // Tell everyone this player got hit

}

}

/\* We 're updating the bullets on the server , so we don 't need

to do this on the client anymore

// Update bullets

for ( var i =0;i< b u l l e t \_ a r r a y . length ;i ++){

var bullet = b u l l e t \_ a r r a y [i];

bullet . sprite . x += bullet . speed\_x ;

bullet . sprite . y += bullet . speed\_y ;

// Remove if it goes too far off screen

if( bullet . sprite . x < -10 || bullet . sprite . x > W O R L D \_ S I Z E . w

|| bullet . sprite .y < -10 || bullet . sprite .y > W O R L D \_ S I Z E . h){

bullet . sprite . destroy ();

b u l l e t \_ a r r a y . splice (i ,1) ;

i - -;

}

}

\*/

}

// H e althBar Logic

**function** c r e a t e H e a l t h B a r () {

meters = game . add . group () ;

// create a plain black r e ctangle to use as the b a c k g r o u n d of

a health meter

**var** m e t e r B a c k g r o u n d B i t m a p = game . add . b i t m a p D a t a (20 , 100) ;

m e t e r B a c k g r o u n d B i t m a p . ctx . be g inPath () ;

m e t e r B a c k g r o u n d B i t m a p . ctx . rect (0 , 0 , m e t e r B a c k g r o u n d B i t m a p .

|  |  |
| --- | --- |
| Creating a simple networked game — Using Javascript | 37 |

width , m e t e r B a c k g r o u n d B i t m a p . height ) ;

m e t e r B a c k g r o u n d B i t m a p . ctx . fi l lStyle = '#000000 '; m e t e r B a c k g r o u n d B i t m a p . ctx . fill () ;

// create a Sprite using the b a c k g r o u n d bitmap data **var** h e a l t h M e t e r B G = game . add . sprite (10 , 10 ,

m e t e r B a c k g r o u n d B i t m a p ) ;

h e a l t h M e t e r B G . f i x e d T o C a m e r a = **true** ;

meters . add ( h e a l t h M e t e r B G ) ;

// create a red r e ctangle to use as the health meter itself **var** h e a l t h B i t m a p = game . add . b i t ma p D a t a (12 , 92) ;h e a l t h B i t m a p . ctx . be g inPath () ;

h e a l t h B i t m a p . ctx . rect (0 , 0 , h e a l t h B i t m a p . width , h e a l t h B i t m a p .

height ) ;

h e a l t h B i t m a p . ctx . fi l lStyle = '# FF0000 ';

h e a l t h B i t m a p . ctx . fill () ;

* create the health Sprite using the red r e ctangle bitmap data

health = game . add . sprite (14 , 14 , h e a l t h B i t m a p ) ; meters . add ( health ) ;

health . f i x e d T o C a m e r a = **true** ;

}

**function** u p d a t e H e a l t h B a r ( h e a l t h R e m a i n ) {

**var**

**var**

**var**

m = ( h e a l t h R e m a i n ) / 100;

bh = 92 - (92 \* m) ;

offset = 92 - bh;

health . key . context . cl e arRect (0 , 0 , health . width , health . height

) ;

health . key . context . fillRect (0 , offset , 12 , bh) ;

health . key . dirty = **true** ;

}

* Displays text that should be shown to the user **function** w r i t e T o H T M L L o g ( message ) {

**if** (( document . g e t E l e m e n t B y I d (" log ") . g e t E l e m e n t s B y T a g N a m e ("li")

. length ) > 6) {

document . g e t E l e m e n t B y I d (" log " ) . in n erHTML = ''; // reset 'Game text '

|  |  |
| --- | --- |
| Creating a simple networked game — Using Javascript | 38 |

}

document . g e t E l e m e n t B y I d ( " log " ) . in n erHTML += "<li >" + message + "

</li > \n" ;

}

**function** u p d a t e b u l l e t s ( bullets ) {

sh o tsLeft . setText (" Shots Left : " + bullets ) ; // console . log( bullets )

}

* D i s c o n n e c t the player and display u n helpful messages **function** gameOver ( player ) {
  + console . log('Is it going here ?');

**var** style = { font : " bold 32 px Arial " , fill : "# fff " ,

b o u n d s A l i g n H : " center " , b o u n d s A l i g n V : " middle " };

* The Text is p o s i t i o n e d at 0 , 100

text = game . add . text ( player . sprite .x, player . sprite .y+50 , "

Game Over " , style ) ;

* text . f i x e d T o C a m e r a = true ;

text2 = game . add . text ( player . sprite .x, player . sprite .y -50 , "

Can 't Shoot no more " , style ) ;

text . se t Shadow (3 , 3 , 'rgba (0 ,0 ,0 ,0.5) ', 2) ;

* text2 . f i x e d T o C a m e r a = true ; player . sprite . addChild ( text ) ; player . sprite . addChild ( text2 ) ;

// We 'll set the bounds to be from x0 , y100 and be 800 px wide

by 100 px high

text . s e t T e x t B o u n d s (0 , 100 , 800 , 100) ;

}

**function** c h e c k O v e r l a p ( spriteA , spriteB ) {

**var** boundsA = spriteA . ge t Bounds () ;

**var** boundsB = spriteB . ge t Bounds () ;

**return** Phaser . Re c tangle . i n t e r s e c t s ( boundsA , boundsB ) ;

}

* if the player fall below certain amount of hp , change the current , likewise if the player regains enough health , show

them in a healthy state

**function** c h a n g e S p r i t e B a s e d O n H e a l t h ( sprite , health ) { **if** ( health >= 70 ) {

**var** c h e c k T h i s O u t = sprite . key ;

**var** s h i p S p ri t e = c h e c k T h i s O u t . replace (/. $/ , "1" )

|  |  |  |  |
| --- | --- | --- | --- |
| Creating a simple networked game — Using Javascript | | | 39 |
| } |  |  |  |
| **else if** ( health < 70 & health > 35) { | | |  |
| **var** c h e c k T h i s O u t = sprite . key ; | |  |  |
| **var** s h i p S p ri t e | = c h e c k T h i s O u t . replace (/. $/ , "2" ) | |  |
| } |  |  |  |
| **else if** ( health | <= 35 & health > 5) { | |  |
| **var** c h e c k T h i s O u t = sprite . key ; | |  |  |
| **var** s h i p S p ri t e | = c h e c k T h i s O u t . replace (/. $/ , "3" ) | |  |
| } |  |  |  |
| **else** { |  |  |  |
| **var** c h e c k T h i s O u t = sprite . key ; | |  |  |
| **var** s h i p S p ri t e | = c h e c k T h i s O u t . replace (/. $/ , "4" ) | |  |
| } |  |  |  |
| **if** ( s h i p Sp r i t e != sprite . key ) { | |  |  |
| console . log ( 'garbo message ') | |  |  |
| sprite . l o a d T e x t u r e ( s h i p S p r i t e ) ; | | |  |
| } |  |  |  |
| **return** sprite |  |  |  |
| } |  |  |  |
|  | Listing 5: Server-side logic | |  |
| **var** express = require ('express ') ; | | // Express contains | some |
| b o i l e r p l a t e to | for routing and | such |  |

**var** app = express () ;

**var** http = require ( 'http ') . Server (app ) ;

* var io = require (' socket . io ')( http ); // Here 's where we include socket . io as a node module

**var** io = require ('socket . io ') . listen ( http ) ;

// Serve the index page

app . get ("/" , **function** ( request , response ) { response . sendFile ( \_\_ d irname + '/ index . html ') ;

}) ;

// Serve the assets d i rectory

app . use ( '/ assets ' ,express . static ( 'assets ') )

app . use ( '/ assets / sound ',express . static ( '/ sound ') )

// load css and js folders

app . use ( '/js ' ,express . static ( \_\_ d irnam e + '/js ') ) ; app . use ( '/ css ' ,express . static ( \_\_ d irname + '/css ') ) ;

// Listen on port 8000

app . set ( 'port ' , ( process . env . PORT || 8000) ) ; http . listen ( app . get ( 'port ') , **function** () {

console . log( 'li s tening on port ' ,app . get ( 'port ') ) ;

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Creating a simple networked game — Using Javascript | | | | | | | | |  |  |  |  |  | 40 |
| }) ; |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **var** | players | = {}; | // Keeps | | | a table of | all | players , | | the key | | | is | the |
|  | socket | id |  |  |  |  |  |  |  |  |  |  |  |  |
| **var** | b u l l e t \_ a r r a y = | | []; | // | | Keeps track | of | all the | | bullets | | to | |  |
|  | update them on the server | | | | | |  |  |  |  |  |  |  |  |
| // Tell Socket . io to start a c cepting c o n n e c t i o n s | | | | | | | | | |  |  |  |  |  |
| **var** | s p a w n O b j e c t s A l l o w e d | | | | = **false** ; // check | | | if the | | server | | is |  |  |
|  | allowed | to spawn health packs and | | | | | d r iftwood | | |  |  |  |  |  |
| **var** | m a x H e a l t h P a c k s = 5; | | | | // there will | | up to 5 health | | | | packs | |  |  |
|  | sc a ttered across the map at any given time | | | | | | | |  |  |  |  |  |  |
| **var** | n u m H e a l t h P a c k s | | = 0; | | // | number of | health packs | | | current | | | on | the |
|  | server |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **var** h e a l t h P a c k \_ a r r a y = []; | | | | | |  |  |  |  |  |  |  |  |  |
| **var** | m a x D r i f t W o o d = 15; | | |  | // there will | | only be 15 drift wood | | | | | | ( |  |
|  | ob s tacles ) s c attered | | | | across the map at any | | | | given | | time | |  |  |
| **var** | n u m D r i f t W o o d = | | 0; | // | number of drift | | | wood | in | the | server | | |  |
| **var** | d r i f t W o o d \_ a r r a y = | | | []; | |  |  |  |  |  |  |  |  |  |

**var** W O R L D \_S I Z E = {w :1500 , h :1000}; // same as the server side world size

io . on( 'c o n n e ct i o n ' , **function** ( socket ) {

* Listen for a new player trying to connect socket . on( 'new - player ', **function** ( state ) {

console . log ( "New player joined with state :" ,state ) ; players [ socket . id] = state ;

* + B r oadcast a signal to everyone c o n t a i n i n g the updated players list

io . emit ( 'update - players ',players ) ;

s p a w n O b j e c t s A l l o w e d = **true** ; // now that a player has connected , allow the server to spawn h e a l t h p a c k s

})

* Listen for a d i s c o n n e c t i o n and update our player table socket . on( 'd i s c o n ne c t ', **function** ( state ) {

delete players [ socket . id ];

console . log(" Player d i s c o n n e c t e d with state : " ,state ) ; io . emit ( 'update - players ',players ) ;

})

* Listen for move events and tell all other clients that so m ething has moved

|  |  |
| --- | --- |
| Creating a simple networked game — Using Javascript | 41 |

socket . on( 'move - player ' , **function** ( p o s i t i o n \_ d a t a ) {

**if** ( players [ socket . id] == u n defined ) **return** ;// Happens ifthe server restarts and a client is still c o nnected

players [ socket . id ]. x = p o s i t i o n \_ d a t a . x;

players [ socket . id ]. y = p o s i t i o n \_ d a t a . y;

players [ socket . id ]. angle = p o s i t i o n \_ d a t a . angle ;

players [ socket . id ]. health = p o s i t i o n \_ d a t a . health ; // change

sprites based on health

io . emit ( 'update - players ',players ) ;

})

* Listen for shoot - bullet events and add it to our bullet array

socket . on( 'shoot - bullet ' , **function** ( data ) {

**if** ( players [ socket . id] == u n defined ) **return** ; **var** n e w \_ b u ll e t = data ;

data . owner\_id = socket . id; // Attach id of the player to the bullet

**if** ( Math . abs( data . speed\_x ) > 20 || Math . abs ( data . speed\_y ) >

1. {
   * console . log (" Player " , socket . id ," is cheating !") ;

}

b u l l e t \_ a r r a y . push ( n e w \_ b u l l e t ) ;

}) ;

})

* Update the bullets 60 times per frame and send updates **function** S e r v e r G a m e L o o p () {

for ( **var** i=0;i< b u l l e t \_ a r r a y . length ;i ++) { **var** bullet = b u l l e t \_ a r r a y [i];bullet .x += bullet . speed\_x ;

bullet .y += bullet . speed\_y ;

* + Check if this bullet is close enough to hit any player for ( **var** id **in** players ) {

**if** ( bullet . owner\_id != id) {

* + - And your own bullet shouldn 't kill you

**var** dx = players [id ]. x - bullet .x;

**var** dy = players [id ]. y - bullet .y;

**var** dist = Math . sqrt (dx \* dx + dy \* dy) ; **if** ( dist < 70) {

io . emit ( 'player - hit ' ,id) ; // Tell everyone this player got hit

console . log (" Player : [" + id + "] , hit by bullet \n" + players [id ]) ;

|  |  |
| --- | --- |
| Creating a simple networked game — Using Javascript | 42 |

}

}

// Check if a player is c o lliding with another player

}

* check if a bullet collides with a heart if so destroy it for ( **var** j=0;j< h e a l t h P a c k \_ a r r a y . length ;j ++) {

**var** dx = h e a l t h P a c k \_ a r r a y [j]. x - bullet .x; **var** dy = h e a l t h P a c k \_ a r r a y [j]. y - bullet .y; **var** dist = Math . sqrt (dx \* dx + dy \* dy) ; **if** ( dist < 40) {

* + io . emit (' player - hit ',id); // Tell everyone this player got hit

h e a l t h P a c k \_ a r r a y . splice (j ,1) ;

j - -;

console . log ( 'Heart number [' + j + '] d e stroyed by bullet '

) ;

numHealthPacks - -;

}

}

* Check if a bullet is c o lliding with d r iftwood for ( **var** j=0;j< d r i f t W o o d \_ a r r a y . length ;j ++) {
  + And your own bullet shouldn 't kill you **var** dx = d r i f t W o o d \_ a r r a y [j]. x - bullet . x; **var** dy = d r i f t W o o d \_ a r r a y [j]. y - bullet . y; **var** dist = Math . sqrt (dx \* dx + dy \* dy) ; **if** ( dist < 40) {
    - io . emit (' player - hit ',id); // Tell everyone this player got hit

d r i f t W o o d \_ a r r a y . splice (j ,1) ; j - -;

console . log ( 'Dr i ftWood number [' + j + '] d e stroyed by bullet [' + i + ']') ;

numDriftWood - -;

}

}

* consider allowing the player to destroy driftwood , maybe if this doesn 't take too long

// Remove the bullet if goes too far off screen

**if** ( bullet . x < -10 || bullet .x > 1500 || bullet . y < -10 ||bullet . y > 1000) {

b u l l e t \_ a r r a y . splice (i ,1) ;

console . log ( 'Bullet [' + i + '] d e stroyed because it went

|  |  |
| --- | --- |
| Creating a simple networked game — Using Javascript | 43 |
| of f screen (' + bullet . x + ',' + bullet . y + | ')' ) ; |
| i - -; |  |
| } |  |
| } |  |
| // check if players collide with hearts |  |
| for ( **var** i=0;i< h e a l t h P a c k \_ a r r a y . length ;i ++) { |  |
| **var** h e a l t h Pa c k = h e a l t h P a c k \_ a r r a y [i]; |  |
| h e a l t hP a c k . x += h e a l t h P ac k . speed\_x ; |  |
| h e a l t hP a c k . y += h e a l t h P ac k . speed\_y ; |  |

// Check if this bullet is close enough to hit any player

* consider spawning " dead " ships that are s h i p w r e c k e d later . for ( **var** id **in** players ) {
  + And your own bullet shouldn 't kill you **var** dx = players [id ]. x - h e a l t h P a c k .x; **var** dy = players [id ]. y - h e a l t h P a c k .y; **var** dist = Math . sqrt (dx \* dx + dy \* dy) ;
* adjust distance based on what will be the final distance between player and ships

**if** ( dist < 40) {

* + - io . emit (' player - hit ',id); // Tell everyone this player got hit

console . log ( 'Co l lision between player [' + id + '] , h e a l t h Pa c k [' + i + ']')

h e a l t h P a c k \_ a r r a y . splice (i ,1) ;

* - -; numHealthPacks - -;

// inform player that health is restored

io . emit ( 'player - heal ' ,id) ; // Tell the player that a heart is found

}

// Check if a player is c o lliding with another player

}

// Remove if it goes too far off screen

**if** ( h e a l t hP a c k . x < -10 || h e a l t h P ac k . x > 1500 || h e a l t h P a c k . y

< -10 || h e a l t h P a c k .y > 1000) {

h e a l t h P a c k \_ a r r a y . splice (i ,1) ;

console . log ( 'H e a l t h Pa c k (' + i + ') destroyed , out of bounds .

(' + h e a l t hP a c k . x + ',' + h e a l th P a c k . y + ')' ) ;

i - -;

numHealthPacks - -;

}

|  |  |
| --- | --- |
| Creating a simple networked game — Using Javascript | 44 |

}

* update the d r iftwood l o cations on the server for ( **var** i=0;i< d r i f t W o o d \_ a r r a y . length ;i ++) {

**var** dr i ftWood = d r i f t W o o d \_ a r r a y [i];dr i ftWood . x += d r iftWood . speed\_x ; dr i ftWood . y += d r iftWood . speed\_y ;

dr i ftWood . rotation += d r iftWood . r o t a t e D i r e c t i o n \* Math . PI / 200; **if** ( dr i ftWood . rotation > 2\* Math . PI) {

dr i ftWood . rotation = 0;

}

* console . log( dr i ftWood . rotation )
* Remove the d r iftwood if goes too far off screen

**if** ( dr i ftWood . x < -10 || d r iftWood . x > 1500 || d r iftWood . y <

-10 || d r iftWood . y > 1000) { numDriftWood - -;

d r i f t W o o d \_ a r r a y . splice (i ,1) ;

console . log ( 'driftWood (' + i + ') destroyed , out of bounds .

(' + d r iftWood .x + ',' + d r iftWood .y + ')' ) ; i - -;

}

}

// Tell everyone where the d r iftwood is:

io . emit ( " driftWood - update " , d r i f t W o o d \_ a r r a y ) ;

// Tell everyone where all the health packs are by sending the whole array

io . emit ( " healthPack - update " , h e a l t h P a c k \_ a r r a y ) ;

* Tell everyone where all the bullets are by sending the whole array

io . emit ( " bullets - update " , b u l l e t \_ a r r a y ) ;

}

* spawn hearts that restore 10 hp when player collides **function** s p a w n H e a l t h P a c k s () {

// Spawn Health Packs for players

**if** ( n u m H e a l t h P a c k s < m a x H e a l t h P a c k s && s p a w n O b j e c t s A l l o w e d == **true** ) {

// spawn health pack

**var** p l u s O r M i n u s x = Math . random () < 0.5 ? -1 : 1;

**var** p l u s O r M i n u s y = Math . random () < 0.5 ? -1 : 1;

**var** n e w \_ h e a l t h P a c k = {x: Math . random () \* 0.75\* W O R L D \_S I Z E . w+50 ,y: Math . random () \*0.75\* W O R L D \_ S I Z E . h+25 , speed\_x : Math . random () \*0.5\* plusOrMinusx , speed\_y : Math . random () \*0.5\* p l u s O r M i n u s y }; h e a l t h P a c k \_ a r r a y . push ( n e w \_ h e a l t h P a c k ) ;

n u m H e a l t h P a c k s ++;

|  |  |  |
| --- | --- | --- |
| Creating a simple networked game — Using Javascript | | 45 |
| console . log ( 'Spawning Health Pack (' + n e w \_ h e a l t h P a c k .x | | + ', |
| ' | + n e w \_ h e a l t h P a c k .y + ')' + n u m H e a l t h P a c k s ) ; |  |
| } |  |  |
| } |  |  |
| **function** s p a w n D r i f t W o o d () { | |  |
| // Spawn D r iftWood as o b st acles players | |  |
| **if** | ( n u m D r i f t W o o d < m a x D r i f t W o o d && s p a w n O b j e c t s A l l o w e d == | **true** |
| ) | { |  |

// spawn health pack

**var** p l u s O r M i n u s x = Math . random () < 0.5 ? -1 : 1;

**var** p l u s O r M i n u s y = Math . random () < 0.5 ? -1 : 1;

**var** ro t ateDir = Math . random () < 0.5 ? -1 : 1;

**var** n e w \_ d r i f t W o o d = {x: Math . random () \* 0.75\* W O R L D \_ S I Z E . w+50 ,

1. Math . random () \*0.75\* W O R L D \_ SI Z E . h+25 , speed\_x : Math . random ()

\*0.5\* plusOrMinusx , speed\_y : Math . random () \*0.5\* plusOrMinusy ,

rotation : Math . random () \*2\* Math .PI , r o t a t e D i r e c t i o n : ro t ateDir }; d r i f t W o o d \_ a r r a y . push ( n e w \_ d r i f t W o o d ) ;

n u m D r i f t W o o d ++;

console . log ( 'Spawning D r iftWood (' + n e w \_ d r i f t W o o d .x + ',' + n e w \_ d r i f t W o o d . y + ')' + n e w \_ d r i f t W o o d . rotation ) ;

}

}

* spawn health packs every 3 seconds ? s e t I n t e r v a l ( spawnDriftWood , 3000) ;
* spawn health packs every 5 seconds ? s e t I n t e r v a l ( spawnHealthPacks , 5000) ;
* 0.016 seconds update

s e t I n t e r v a l ( ServerGameLoop , 16) ;